Kaiserslautern Mainframe Summit

Kaiserslautern Mainframe Summit

Mainframes Today

Seminar Proceedings

Motivation des Summits

Mainframes bilden auch weiterhin das Rückgrat der IT in vielen Organisationen. DozentInnen der IBM werden in Kooperation mit Lehrenden des Fachbereichs Informatik Architektur und Einsatzmöglichkeiten von Mainframes in modernen IT-Landschaften in Vorträgen und Übungen vermitteln.

Seminarinformation

Anschließend an den erfolgreichen KMS, mit 40 Teilnehmern, fand das Seminar im WS 2010/2011 der AG DBIS und AG HIS statt. Dabei entstanden 13 Arbeiten passend zu den Vortragsthemen. Diese Seminararbeiten sind in diesem Band zusammengefasst.

Allgemein Mainframes und z/OS

1. Die Geschichte des Mainframes
2. Hardware of an IBM Mainframe zEnterprise 196
3. Redundanz
4. Virtualisierungstechnologien
5. Heterogene Umgebungen von Mainframes
6. An Overview over the basic z/OS Interfaces
7. Jobmanagement in z/OS
8. Programming languages on z/OS
9. Working with z/OS: Applications
10. IBM-Zertifizierungen

DB2 for z/OS

11. Data Sharing in einer z/OS-DB2 Umgebung
12. An Overview of the Support for XML in the DB2 Database System
13. Schnelles Klonen

© 2011, Karsten Schmidt (TU Kaiserslautern), kschmidt@cs.uni-kl.de

Technische Universität Kaiserslautern
DBIS Datenbanken und Informationssysteme
HiS
IBM

Die Geschichte des Mainframes

Sebastian Bock[1], Vasil Tenev[1]

[1] Technische Universität Kaiserslautern,
Fachbereich Informatik, Mainframe Summit,
Erwin-Schrödinger-Str. Gebäude 48
67663 Kaiserslautern, Deutschland
{s_bock, v_tenev}@cs.uni-kl.de

Abstrakt. Als Hauptthersteller von Großrechnern verzeichnet IBM die Meilensteine in der Geschichte des Mainframes. Eine Einführung gibt uns den Überblick über die Fortentwicklung der Rechnermaschinen. Hier werden die wichtigsten Erfindungen vorgestellt. In folgenden Abschnitten kann der Leser die Entwicklung des Mainframes von IBM über die Jahre von 1964 bis 2010 verfolgen und sich mit einigen interessanten Fakten aus der Geschichte bekannt machen. In dieser Seminararbeit wurde der Schwerpunkt auf die Fortschritte gelegt, die das Mainframe von System/360 bis zum System z führten.

Stichworte: OS, Models, Installations, Protocols (VTAM, SNA, TCPIP, ...), basic concepts (e.g., LPAR), "zero" downtime, EBCDIC, crypto/security features, high availability, scalability

1 Einführung

Die Geschichte der Großrechner führt uns ins Jahr 1500, als Leonardo da Vinci den ersten mechanischen Rechner entwarf. Langsam wurde die Technologie weiterentwickelt. Nach über 350 Jahre um 1880 konstruierte Herman Hollerith das Lochkarten-System, um die Volkszählung in den USA zu automatisieren. Seine Erfindung wurde später an eine Firma verkauft, die heute IBM heißt. [1]

1.1 Die erste Generation – Die Röhren-Großrechner

Der amerikanische Physiker Lee De Forest erfand 1906 die Vakuumröhre und damit machte er den ersten Anstoß in der Technologie in Richtung des Rechners. Erst 3 Jahrzehnte später wurde der erste elektronische Digitalrechner von Dr. John V. Atanasoff und seinem Assistent Clifford Berry gebaut. Ihre Maschine, der *Atanasoff-Berry-Computer* (ABC), bildete die Grundlagen für die Fortschritte in den elektronischen Digitalrechnern. Der ABC wurde in den nächsten fünf Jahren von einer ganzen Reihe leistungs- und funktionsfähigerer Maschinen gefolgt.

1941 Konrad Zuse stellte den ersten programmierbaren Computer *Z3* vor. Im zum Lösen komplexer Ingenieur-Gleichungen entworfenen Z3 wurde eine binäre Darstellung anstelle des Dezimalsystems angewendet.

1942 *British Intelligence's Colossus* wurde vom britischen Mathematiker Alan Turing während des 2. Weltkrieges gebaut, um die Enigma-Verschlüsselung zu knacken. Der Koloss war eine large-scale elektronische Maschine, die im Vergleich zu den bisherigen Rechnern logische Operationen anstelle von arithmetischen durchgeführt hatte.

1944 Der über dreißig tonige *ENIAC* (electronic numerical integrator and calculator) wurde für ballistische Berechnungen benutzt und spielte eine Rolle beim Entwurf der Atombombe. Mit seinen 19.000 Vakuumröhren, 1.500 Relais, Hunderttausenden von Widerständen, Kondensatoren, Induktoren und einem Verbrauch elektrischer Leistung von fast 200 Kilowatt war ENIAC ein Prototyp für die meisten modernen Rechner.

1944 Die IBM Ingenieure konstruierten in Kooperation mit Howard Aiken den *Harvard Mark I*. Dieser Automatic Sequence Controlled Calculator (ASCC) konnte mit allen vier arithmetischen Operationen umgehen und hatte spezielle eingebaute Programme für logarithmische und trigonometrische Funktionen.

Im nächsten Jahr (1945) änderte sich die Richtlinie der Entwicklung weg von den Lochkarten dank John von Neumann und seines Berichtes „First Draft of a Report on the ENVAC [Electronic Discrete Variable Automatic Computer]“ über die Architektur des speicherprogrammierbaren Rechners. [1]

1.2 Die Halbleiter-Revolution

Am 23. Dezember 1947 wurde von William Shockley, Walter Brattain, und John Bardeen aus den Bell Laboratorien der erste Germanium-Spitzentransistor vorgestellt. Die neue Technologie steuert zur Reduktion der elektrischen Leistung und der Größe von den Rechnermaschinen bei. Vier Jahre später kam der erste hergestellte kommerzielle Großrechner *UNIVersal Automatic Computer I*. UNIVAC I konnte fast 100.000 Additionen zweier 10-stelliger Zahlen innerhalb einer Sekunde durchführen. Der erste von diesen walk-in Computern mit Größen einer Ein-Auto-Garage wurde an die Volkszählungsbehörde der Vereinigten Staaten ausgeliefert. [1]

Der erste Mainframe *IBM 701* wurde vom Präsident des Unternehmens Thomas J. Watson, Jr. am 29. April 1952 vorgestellt. Auf dem jährlichen Treffen der Aktionäre wurde der neue Rechner als “der modernste, flexibelste High-Speed-Computer der Welt” (Watson) bezeichnet. Der Rechner bestand aus zwei Band-Einheiten, einem magnetischen Trommelspeicher, eine Kathodenstrahlröhre-Speichereinheit, einer Lagearithmetik- und Speichereinheit mit einem Operator-Panel, einem Kartenleser, einem Drucker, einem Card-Punch und drei Energieeinheiten. [2]

Die Maschinen der IBM 700 Serie wurden während der nächsten Jahre immer leistungsfähiger. Die im Jahr 1955 vorgestellten IBM 704 und IBM 705 sind nur zwei Beispiele dafür. *IBM 704* trug mit Gleitkomma-Arithmetik-Befehlen und der Anwendung von FORTRAN in einem Großrechner zur Lösung komplexer wissenschaftlicher, technischer und geschäftlicher Probleme bei. Danach kam der an der Verarbei-

tung geschäftlicher Daten orientierte *IBM 705* mit 400 Multiplikationen zweier Zahlen in Größenordnung von einer Milliarde pro Sekunde auf den Markt. [3]

Abbildung 1 704 Data Processing System

Mit der Zeit wurden immer weniger Vakuumröhren beim Entwerfen von neuen Rechnerarchitekturen benutzt und das brachte den Computer in die Epoche der Transistoren.

1.3 Die Transistor-Rechnersysteme

Weniger Wärme, kleinere Massen und größere Zuverlässigkeit waren die Hauptvorteile eines Transistor-Rechners. 6 Jahre später, nachdem der erste experimentelle

Transistor-Rechner an der Universität Manchester erfolgreich in November 1953 gebaut wurde [4], wurden die ersten volltransistorierten *IBM 7090* Computer-Systeme verkauft. Das neue Modell der IBM 700 Serie war fünffach schneller als sein Vorgänger und blieb noch 10 Jahre in Produktion. [5]

Das nächste Produkt von IBM war über 1,5 Meter hoch und fast einen Meter breit. *IBM 1401* war der erste in Serie gefertigte, digitale, volltransistorierte Businessrechner, der an viele Unternehmen weltweit geliefert wurde. Gegen eine monatliche Miete von $2.500 (und höher, je nach Konfiguration) bekam man einen Großrechner, mit dem 193.300 Additionen achtstelliger Zahlen oder 25.000 Multiplikationen von sechsstelligen mit vierstelligen Zahlen in einer Minute möglich waren. Ende 1961 erreicht die Anzahl installierter Rechner allein in den USA 2.000 – was einem Viertel aller eingesetzten elektronischen speicherprogrammierbaren Computern von allen Herstellern entspricht. Mitte der 1960er Jahre erreicht das System seinen Höhepunkte mit mehr als 10.000 Installationen und beherrschte den Markt bis Februar 1971. [1]

Auf IMB 7090 folgte das IBM 7094 Data Processing System mit 1,4-bis-2,4-facher Leistung je nach Applikation. Die neue Maschine war kompatibel mit seinem Vorgänger, hatte eine erheblich höhere interne Geschwindigkeit und war so der gestiegenen wissenschaftlichen Auslastung der Systeme in den 1960er Jahren gewachsen. IBM produzierte den Großrechner von Januar 1962 bis Juli 1969. Der typische 7094 wurde für $3.134.500 mit komplettem Softwarepaket von 7090/7094-Programmen inklusive der Programmiersprachen FORTRAN und COBOL, Input/Output-Steuerung und Sortierung ohne zusätzliche Kosten verkauft. [1]

1.4 Multiprocessing und Betriebssysteme

Bevor System/360 von IBM auf die Welt kam, wurde 1961 das erste Rechnersystem mit Hardware gesteuertem Kellerspeicher und mit für extensive Nutzung ausgelegte Deskriptoren für den Datenzugriff entwickelt. *Burroughs B5000 Mainframe* bot nun die Unterstützung von Multiprogrammieren und Multiprocessing an. Mit diesem Großrechner wurde der erste Schritt in die neue Entwicklungsrichtung gemacht. Ab jetzt ist das Multiprocessing nicht nur ein wichtiges Thema für die Rechnerherstellerindustrie, sondern auch ein Hauptmerkmal jedes Computersystems. [1]

2 Über 45 Jahre IBM Mainframe

In nächsten Abschnitten verfolgen wir die Verwandlung des IBM Großrechners vom System/360 bis System z. Dabei werden wir unsere Aufmerksamkeit auf die wichtigsten Meilensteine dieses Entwicklungsprozesses aus den Themen der Hardware und Betriebssysteme mit deren Haupteigenschaften (Grundkonzepte, Protokolle, Sicherheitsfeatures, Verfügbarkeit, Skalierbarkeit und andere) schenken.

2.1 System/360

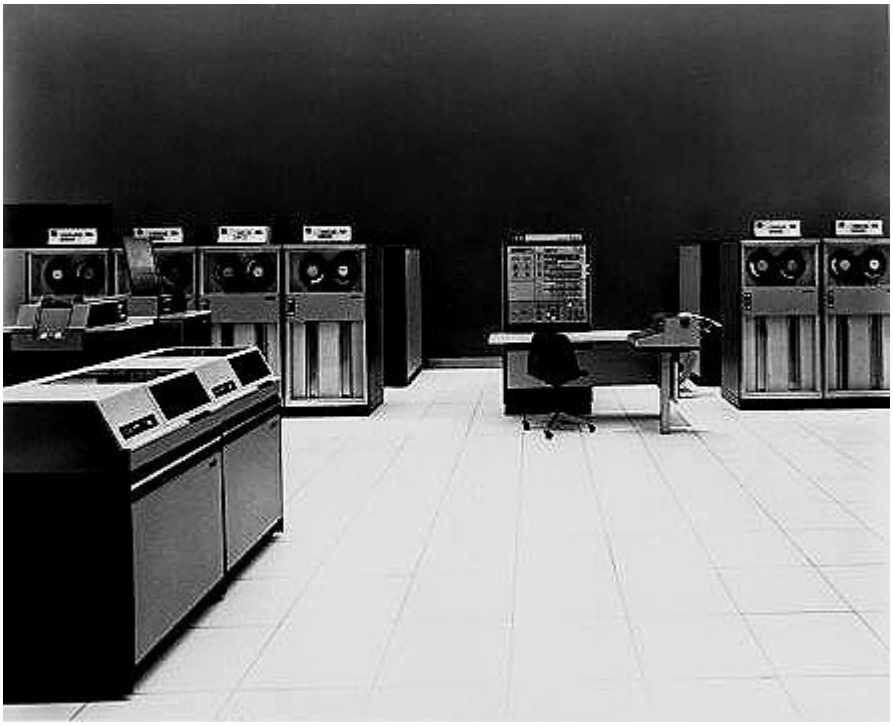


Abbildung 2 Eine System/360 Installation

Das am 7. April 1964 vorgestellte System/360 besitzt eine völlig neue Architektur, die speziell für Datenverarbeitung als auch für Kompatibilität in einem breiten Spektrum von Leistungsniveaus ausgelegt wurde. Die traditionelle Unterscheidung zwischen Rechnern für kommerzielle und wissenschaftliche Nutzung war in System/360 eliminiert. Benutzer konnten geschäftliche und wissenschaftliche Probleme oder eine Kombination der beiden Verfahren mit gleicher Wirksamkeit erledigen. Mit der gleichen Art von Input/Output-Geräten konnte der Benutzer sein System/360 in jedem Punkt im Leistungsbereich erweitern, ohne es umprogrammieren zu müssen. Diese Vielseitigkeit wurde durch die Vielzahl von Peripheriegeräten verstärkt. IBM investierte fünf Milliarden Dollar, um die System/360 Familie mit fünf leistungsfähigen Rechnern zu entwickeln. Auf diesen lief das gleiche Betriebssystem und alle konnten die 44 Peripheriegeräte mit gleicher Architektur nutzen, Informationen speichern und im System einfügen oder abrufen lassen.

Mikroelektronische Schaltungen – Produkt von IBM Solid Logic Technology – bildeten die grundlegenden Baugruppen des System/360, welches das erste kommerziell verfügbare Datenverarbeitungssystem war, deren Design auf dem Einsatz von mikrominiaturisierten Rechnerschaltungen basiert. Hinter diesen Schaltungen verbirgt

sich das Konzept von Integration und Miniaturisierung, also immer mehr Bauteile zu einem zusammenzufassen und diese immer kleiner zu machen.

Eine Hierarchie von Speichern des Systems/360 ermöglichte den Zugriff auf Daten aus dem Kernlager in unterschiedlichen Geschwindigkeiten. Kleine lokale Datenspeicher operierten mit 200 Milliarden Zugriffen pro Sekunde. Die Steuerungsspeicher besaßen Zugriffszeit von 250 Nanosekunden und die leistungsfähigen Speicher von 2,5 bis 1 Nanosekunde. Die Kerndatenspeicherkapazität reichte je nach Bedarf von 8.000 bis 8.000.000 Zeichen. Dies sind über sechzig Mal mehr direkt adressierbare Zeichen als bisher in IBM-Rechnern.

Eingebaute Kommunikationsfähigkeiten machten System/360 für entferne Terminals unabhängig von der Distanz verfügbar. Hunderte von Endgeräten konnten gleichzeitig mit einem System kommunizieren, während der Rechner weiterhin die grundlegende Aufgabe des Arbeitsprozesses ausführte. Die Ausstattung wurde durch Programme unterstützt, die System/360 das Planen seiner eigenen Aktivitäten zum Erreichen von Non-Stop-Computing ermöglichen. [6]

2.1.1 EBCDIC

Hinter dieser Abkürzung verbirgt sich eine 8-Bit-Zeichenkodierung und steht für Extended Binary Coded Decimal Interchange Code. Wie der Name schon sagt, ist es eine Erweiterung der bis dahin genutzten numerischen Kodierung BCD, die Benutzern den Austausch von Zahlen zwischen verschiedenen Computern mithilfe von Lochkarten ermöglichte. Mit dem Anfang der 1960er Jahre von IBM entwickelten EBCDIC kam zu den Zahlen auch noch Groß- und Kleinbuchstaben hinzu, sodass nicht nur Text-Daten sondern sogar Programme getauscht werden konnten.

Die Kodierung besteht aus acht Bit (also einem Byte), welche in zwei Nibbles mit jeweils vier Bits eingeteilt sind. Das erste Nibble steht für Zeichenklasse (z.B. 1111 für Zahlen) und das zweite für das spezifische verschlüsselte Zeichen. [7] Es sind nicht alle 256 ($=2^8$) Kombinationen belegt.
1964 wurde der erste Mainframe, das System/360, mit EBCDIC als interner Verarbeitungscode auf den Markt gebracht und mit dem schnellen Erfolg des /360 verbreitete sich auch der EBCDIC. Drei Jahre danach wurde die heute eher bekannte ASCII-Kodierung standardisiert. Die ASCII-Popularisierung dieser 7-Bit-Zeichenkodierung führte dazu, dass EBCDIC sich außer auf Mainframes nie durchsetzten konnte und die Mainframes auch mit diesem Zeichensatz zurechtkommen mussten. [8] Die dafür notwendige Übersetzung findet heute im z/196 auf Prozessorebene statt. Außerdem speichert der z/OS Web Server Daten im ASCII-Format, da die gängigen Webbrowser auf PCs mit ASCII-Daten arbeiten. Das native Linux läuft auch auf ASCII, Java auf UNICODE. [9] Aber EBCDIC wird auch heute noch auf den Mainframes genutzt, was vor allem auch auf die Rückwärtskompatibilität zurückzuführen ist.

2.1.2 Betriebssystem OS/360

Ursprünglich wurde ein Batch-orientiertes Betriebssystem OS/360 von IBM geplant. Warum sich später in der Produktion zu einem einfacheren Betriebssystem entschlossen wurde, dafür gab es mindestens zwei Gründe. Einerseits passte das OS/360 nicht in den begrenzten Speicher auf den kleineren System/360 Modellen und IBM wurde andererseits bewusst, dass die Entwicklung der beiden Komponenten OS/MFT und OS/MVT, die später noch genauer erklärt werden, viel länger als erwartet dauern würde. Dies lässt sich auf die gestiegene Komplexität des System/360's Betriebssystems zurückführen. Es musste Multiprogramming unterstützen, weil die schnelleren CPUs sonst die meiste Zeit auf I/O-Operationen warten würden und allen gültigen Anwendungen alle beantragten Dienste zur Verfügung stellen. Hinzu kommt eine stabile Behebung von Abstürzen oder Fehlverhalten einer Applikation, ohne die zur gleichen Zeit ausgeführten Anwendungen anzuhalten. Eine viel breitere Palette von Baugrößen sollte unterstützt werden (Speichervolumen von 16 KB bis 1 MB und Prozessorgeschwindigkeiten zwischen ein paar Tausend und 500.000 Instruktionen pro Sekunde). System/360 Betriebssysteme mussten eine Vielzahl von Anwendungen unterstützen, die sich durch unterschiedliche Datenzugriffe auszeichneten: manchmal sequentiell von Anfang bis Ende, ein anderes Mal schnell und direkt auf bestimmte Datensätze in sehr großen Dateien und manche stellten große zeitaufwendige Berechnungen mit sehr wenigen Schreib- und Lesezugriffen an. [10]

Zum Entwickeln und Testen der geplanten Betriebssysteme war die Verwendung von System/360-Hardware erforderlich. So wurde zunächst **Basic Programming Support (BPS)** entwickelt. Mittels BPS sind dann die zum Entwickeln von DOS/360 und OS/360 benötigten Dienste und bereitstehende Werkzeuge entstanden, wie Compiler (FORTRAN und COBOL), Sortieren und vor allem der Assembler. [10]

Danach brachte IBM **Basic Operating System/360 (BOS/360)** heraus, das aus einem Kartenleser geladen wurde. Das System unterstützte Bandlaufwerke und einige Festplatten. Für IBM 1401 Rechner mit Bandlaufwerken aber ohne Festplatten wurde als Upgrade-Pfad das **Type Operating System/360 (TOS/360)** entworfen. [10]

Als nächstes wurde von den Entwicklern des BOS/360 und TOS/360 das **Disk Operating System/360 (DOS/360)** gebaut. DOS/360 ist zu einem Mainstream-Betriebssystem geworden, dessen Nachkomme z/VSE noch weit verbreitet ist. [10]

Später mit der Ankündigung von System/360-67 wurde das Timesharing- Betriebssystem **TSS/360** vorgestellt, das die neuen Virtual-Memory-Fähigkeiten des Modells 67 ausnutzten sollte. Allerdings dauerte die Fertigstellung von TSS/360 zu lange und es war dann auch noch so unzuverlässig, dass IBM es nie offiziell verkaufte. Bis zu diesem Zeitpunkt war das von IBM Cambridge Scientific Center entwickelte alternative **CP-67** gut genug, um sie ohne Garantie als Timesharing Facility an wenige Großkunden anzubieten. Auf CP-67 folgte VM/370 und schließlich z/VM. [10]

PCP (Primary Control Program) war eine sehr frühe Version von OS/360, das Multiprogramming nicht unterstützte. OS/360 entpuppte sich nicht als Betriebssystem, sondern als eine Bezeichnung für eine Gruppe von Betriebssystemen. PCP war eine Notlösung, die nur ein Programm zu einem Zeitpunkt ausführen konnte, daher wurde ab Mitte der 1960er Jahre **OS/MFT ("Multiprogramming with Fixed num-**

ber of Tasks") und **OS/MVT ("Multiprogramming with Variable number of Tasks")** verwendet. [10]

OS/MFT und OS/MVT boten sehr ähnliche Features, aber hatten unterschiedliche Ansätze zur Speicherverwaltung. Bei der Installation von OS/MFT, konnten die Kunden bis zu fünf Partitionen des Arbeitsspeichers mit festen Größen spezifizieren, in denen Applikationen gleichzeitig ausgeführt werden konnten. OS/MVT behandelte den gesamten nicht vom Betriebssystem verbrauchten Speicher als einen einzigen Pool, aus dem konnten benachbarte Speicherzellen allokiert werden, die von beliebig vielen, gleichzeitig laufenden Anwendungsprogrammen angefordert wurden. Diese Regelung war flexibler als OS/MFT und verwendete den Speicher im Prinzip effizienter, führte aber zu Fragmentierung. Obwohl es genügend freien Speicher insgesamt gab, war dieser in einzelne kleine Stücke geteilt, die nicht groß genug waren, um ein Programm auszuführen. [10]

System/360 mit einer monatlichen Miete ab $2.700 für eine Grundkonfiguration und bis zu $115.000 für eine Multisystem-Konfiguration typischer Größe besaß auch wirtschaftliche Vorteile. Die vergleichbaren Kaufpreise von Konkurrenzprodukten lagen zwischen $133.000 und $5.500.000. [6]

Über die Anzahl der installierten Systeme über die Jahre lässt sich nur schwerlich etwas finden. Die einzigen gefunden Zahlen zeigen zwischen 1964 und 1969 einen enormen Anstieg der Installationen von 17.000 auf 90.000. [11]

Das System hatte den Anspruch allumfassend zu sein. Daher wählte man die Zahl 360 als Hinweis auf den maximalen Grad eines Winkels. Der Anspruch konnte allerdings nicht erfüllt werden, sodass im Nachhinein die Zahl anders definiert wurde. „3“ wurde als IBM-Standard gewertet und „60“ für ein in den 1960ern entwickeltes Produkt. Daher hießen die Nachfolger System/370 und System/390. [12]

2.2 System/370

Im Juni 1970 machte IBM den nächsten großen Schritt und stellte die Hardware-Architektur System/370 vor. Die verbreitesten Modelle waren 370/155 und 370/165. In den Jahren zuvor wurde in Sachen Speicher weiterhin das Konzept der Integration und Miniaturisierung verfolgt und so war die Forschung und Entwicklung jetzt soweit vorangeschritten, das man in der Lage war alle gleichen Elemente des Speichers (Widerstände, Kondensatoren und Dioden) auf eine einzige Siliziumscheibe zu pressen. Diese Speicher nannte man Monolithische Speicher und später RAM (Random Access Memory). Im System/370 wurden nun 128 bit Bipolartransitoren eingesetzt, die zu einer erheblichen Leitungssteigerung führten. Die Speichergröße stieg damit auf 32-512 kB und eine Rechnerleistung von 0,08 MIPS [13] wurde erreicht (abgerechnet wurde pro MIPS: ca. 1,25 Mio. €/MIPS).

Mit der Abkehr von Magnetkernspeichern, die bis dahin die Hauptstütze der Rechenspeicher darstellten, hinzu Halbleitern war entscheidend für das System/370. In diesem war es später möglich 64K, 288K oder sogar 1-Megabit Speicherchips einzusetzen.

Revolutionär war die vollständige Virtualisierung der Mainframe-Hardware, die sich mithilfe dynamische Adressumsetzung (DAT) in logische Partitionen, so genann-

Abbildung 3 System/370 Model 145

te LPARs, aufteilen lies. Eine neue Art der Betriebssysteme war geboren: die Familie der Virtuellen Maschinen (VM). Das Herz der VMs ist der Hypervisor. Er hatte dabei die Aufgabe der Koordination [14]: Er wies verschiedenen installierten VMs die gerade benötigenten Ressourcen zu oder zog nicht benötigte ab und war in der Lage jedem Benutzer bis zu 16 MB virtuellen Speicher zur Verfügung zu stellen.[15] Stürzte eine Maschine ab, wurde der Betrieb anderer davon nicht beeinträchtigt.

Aufgrund dieser Neuerungen war nun ein stabiles TSO möglich. TSO ist die Abkürzung für Time-Sharing Option. Sie ermöglichte den Mehrbenutzerbetrieb auf einem Rechner, d.h. jeder Benutzer bekommt für eine bestimmte Zeit einen Teil der CPU zugewiesen, währenddessen andere Benutzer mit ihrer Transaktion warten müssen. [16]

Für all diese Neuerungen wurden zwischen 1971 und 1972 die Betriebssysteme S/VS1 und OS/VS2 SVS (single virtual storage) eingeführt. Sie ersetzten MFT und MVT vom System/360, deren größtes Problem in der Fragmentierung des Speichers in zu kleine Einheiten bestand. Durch die Einführung von virtuellem Speicher konnten jetzt größere Speicherbereiche allokiert werden, als physisch eigentlich vorhanden. Das grundlegende Problem blieb erhalten, hatte aber jetzt nicht mehr so große Auswirkungen. Gelöst wurde es mit dem schnellen Nachfolger MVS im Jahr 1974. [17] Es erlaubte beliebig viele Programme in unterschiedlichen Adressbereichen auszuführen und brachte eine Reihe von Zusatzfunktionen mit.

Zunächst wäre da der RMF, Resource Measurement Facility, zu nennen. Er dient als Performance Monitor, damit das System aktuell auf die Bedürfnisse/Geschehnisse reagieren kann. Deutet sich zum Beispiel eine baldige Arbeitsaufwandsteigerung an, kann der RMF dies rechtzeitig anzeigen und damit frühzeitig darauf reagiert werden.[18]

Das nächste Tool diente für einen Bereich, der von Anfang an sehr wichtig war, dessen Anforderungen sich aber auch extrem verändert haben: die Sicherheit. Allein

die Bedeutung des Begriffs Sicherheit hat sich stark verändert. Zunächst verstand man darunter, dass der Mainframe zuverlässig funktioniert. Der Datenschutz war daher gegeben, dass kaum jemand wusste wie man auf die Daten zugreifen konnte, weil das technische Wissen nur wenigen Menschen vorbehalten war. Bis heute ist das Computerfachwissen und auch die Anzahl der Leute, die sich damit auskennen, aber enorm gestiegen. Somit müssen Daten viel besser geschützt werden und zwar gegen Angreifer von außen und innen. Das Sicherheitstool RACF, Resource Acces Control Facility, schützt die kritischen Systemressourcen und kontrolliert was ein Benutzer darf und was nicht. Die drei Hauptfunktionen dafür sind Identifikation und Verifikation der Benutzer mittels Benutzerschlüssel und Passwortprüfung (Authentifizierung), Schutz von Ressourcen durch die Verwaltung der Zugriffsrechte (Autorisierung), Logging der Zugriffe auf geschützte Ressourcen (Auditing).[19]

Ein weiteres Werkzeug war das JES (job entry subsystem). Die Computersysteme wurden immer stärker verbunden. Man bildete sogenannte Cluster um Rechenaufgaben/Jobs auf mehrere Knoten zu verteilen und parallel auszuführen. Die Aufteilung übernimmt das JES, indem es Jobs einem Betriebssystem zuführt und die berechneten Ergebnisse entgegen nimmt.[20]

Mitte der 70er führt IBM auch ein Netzwerkprotokoll SNA (Systems Network Architecture) ein. Dabei handelte es sich um einen kompletten Protokollstack, der hierarchisch und zentral organisiert war. Mainframesysteme konnten darüber telekommunizieren und das Protokoll umfasste standardmäßig schon einige Services für die Übertragung von Daten oder den Durchgriff von Terminals und Druckern über andere Systeme (*Passthrough*).[21] IBM wollte eine Alternative zu den teuren Batchsystemen geben und erhoffte sich so mehr Hardwareumsatz. Die Software die dafür auf den Mainframes installiert wurde heißt VTAM: Virtual Telecommunications Access Method. Später wurde der Versuch unternommen mit APPN (Advanced Peer-to-Peer Networking) ein Peer-to-Peer Protokoll durchzusetzen. Die Migration von SNA zu APPN erwies sich als schwierig und außerdem begann dann sich das dezentral organisierte TCPIP durchzusetzen.

In den 80ern wurde das System noch weiter ausgereizt. Die 31-bit-Adressierung wurde eingeführt. Das Problem der statischen Konfiguration der I/O-Geräte wurde mithilfe der Dynamic Channel Architecture gelöst. Je nach Anforderungen konnte man auf bis zu acht Channels pro Gerät verfügen, was die Erreichbarkeit und Effizienz stark verbesserte. [22] 1983 wurde ein eigenes relationales Datenbankensystem von IBM entwickelt, das DB2.

Die typischen Preise für ein System/370 lagen zwischen 2,2 und 4,6 Millionen Dollar. Zusätzlich gab es ja dann noch die Möglichkeit für den IBM Service, der nochmal zwischen 6000-12000 Dollar pro Monat kostete. [23]

Die immensen Kosten schreckten immer mehr Kunden ab. Zusätzlich setzte sich das Konzept der Dezentralisierung mit Client-Server-Architekturen immer mehr durch.

2.3 System/390

Anfang der neunziger Jahre wurde das Konzepts des Mainframes eigentlich für nicht zukunftsfähig erklärt. IBM musste sich neu erfinden und brachte ein weitumfassendes neues System auf den Markt, bei dem eine neue billigere Preispolitik verfolgt wurde, die vor allen Dingen auf den Support ausgelegt war. Mit der Umstellung auf Luftkühlung in 10 von 18 neuen Systemen und einer verbesserten Energieeffizienz wurden somit auch die Unterhaltkosten für den Kunden gesenkt. Außerdem wurde nun auch die Unterstützung von offenen Standards und Betriebssystemen wie Linux ermöglicht. Zusammen mit dem großen Interesse der Kunden an einem hohen Maß an Sicherheit und Rechenleistung führte dies dazu, dass der Mainframe bis heute nicht ausgestorben ist.

Das System/390 wurde 1990 angekündigt und hatte die Modellnamen G1 bis G6. In den ersten zwei Generationen wurden Bipolartransistoren später dann die komplementärer Metall-Oxid-Halbleiter (CMOS) als Speicher- und Logikeinheit verwendet. Diese benötigen weniger Energie und Platz und sind zusätzlich auch noch kostengünstiger als Bipolartransistoren.

Der Systemspeicher war nicht mehr auf 2 GB limitiert. Ein einzelner Adressbereich konnte höchstens 31 Bit (also 2 GB) haben, aber es war möglich mehrere gleichzeitig zu einem “Expanded Storage” zu konfigurieren. Diese Speicherbereiche konnten für extrem schnelles Paging, Disk-Caching oder virtuellen Speicher für VMs benutzt werden. [24]

Es bleibt zu erwähnen, dass auch dieses System rückwärtskompatibel zu den vorhergehenden Modellen war. Außerdem brachte es schnelle Verschlüsselungstechniken direkt auf Prozessorebene mit. [25]

Die neue Enterprise System Connection (ESCON) Architektur erlaubte dem Kunden seine Hardware in einem Radius von 9 km zu verteilen. So ließ sich jetzt auch das Konzept des Parallel Sysplex umsetzten. Hierbei handelt es sich um ein Mainframecluster, das Datenaustausch und parallele Verarbeitung betreibt und das vor allen Dingen zur Notfallwiederherstellung aufgebaut wird. Es setzt sich aus den folgenden Komponenten zusammen: Zunächst wäre da die Coupling Facility (CF) Hardware, die verschiedenen Prozessoren Zugriff auch die gleichen Daten ermöglicht und für eine flexible Lastverteilung und Skalierung sorgt. [26] Hinzu kommt das Sysplex Timers oder Server Time Protocol, das die Zeitsynchronisation aller beteiligten Systeme regelt, und die schon erwähnte schnelle und qualitative Verkabelung. Außerdem ist natürlich auch Software notwendig und zwar im Betriebssystem als auch auf der Middleware-Schicht (DB2). [27]

Mit diesen Funktionalitäten befand sich der Mainframe auf dem Weg zur Hochverfügbarkeit, der es ermöglichen sollte 24 Stunden am Tag verfügbar und ohne Ausfall zu arbeiten. Diese konnte aber beim System/390 nur wenig mehr als 99,9% erreichen, was aufs Jahr hochgerechnet ungefähr 9 Stunden Ausfallzeit entspricht.
Die Preisunterschiede für luft- und wassergekühlte System war sehr groß, was natürlich mit den unterschiedlichen Leistungsfähigkeiten zu tun hatte. Aber es war jetzt möglich sich einen luftgekühlten Mainframe für 70.500 Dollar zu kaufen. Natürlich

gibt es nach oben keine Grenzen und man konnte auch 3,12 Millionen ausgeben. Wassergekühlte Mainframes kosteten zwischen 2,45 und 22,8 Millionen Dollar.

2.4 IBM System z

Gegen Ende der 90er wurden immer mehr Geschäftsprozesse eines Unternehmens mithilfe von Informations- und Kommunikationstechnologien automatisiert (eBusiness) [28]. IBM entwickelte seinen Mainframe weiter um seine Kunden auch hier unterstützte zu können und ermöglichte die Ausführung von tausenden Servern auf einem Mainframe. Mitte 2000 wurde die Serie z gestartet. z steht hierbei für "zero downtime". Damit ist klar, worum es geht: Hochverfügbarkeit. IBM wirbt mit Verfügbarkeitsklassen von 99,999%, was höchstens fünf Minuten Totalausfall im Jahr entspricht. Dazu ist die Hardware von System z komplett redundant ausgelegt, z.B. existiert sogar jede Kabelverbindung doppelt. [29]
Das System z zeichnet sich vor allen Dingen durch eine 64-Bit-Adressierung auf physischer und virtueller Ebene aus. Dies löst endlich die Zwänge der 31-bit-Hardware und so etwas wie "Expanded Storage" gibt es jetzt nicht mehr. Trotzdem unterstützt das System auch weiterhin 24-bit und 31-bit-Adressierung und zwar durch verschiedene Modi für jede Adressierungsart. Als Verschlüsselungstechnik wird jetzt der Advanced Encryption Standard (AES) eingesetzt, der auch in 802.11 (WLAN) zur Verwendung kommt.

Abbildung 4 eServer zSeries 990 [30]

In Abbildung 4 ist ein z900 abgebildet. Er hat die Größe eines Kleiderschranks. In diesem sind keine Festplatten integriert, solche können durch ESCON angebunden werden. Dafür lassen sich in ihm bis zu 32 zentrale Prozessoren installieren. Diese können sogar einzeln für spezielle Aufgaben konfiguriert werden. Durch Sysplex-Mechanismen lassen sich bis zu 32 von diesen Frames verbinden und dabei kann jeder bis zu 100 Kilometer voneinander entfernt stehen. [31]

Ein neues Betriebssystem wurde damit auch auf den Markt gebracht. Das neue z/OS lässt sich parallel zu anderen Betriebssystemen (Linux in zSeries, z/VM) ausführen.

Vor kurzem wurde der z196 angekündigt. IBM scheint damit auf einen guten Weg zu sein auch in Zukunft mit Mainframes Geld zu verdienen. Von Anfang an wurde das System kontinuierlich weiterentwickelt und war des Öfteren Vorreiter in Funktionalität und Ausstattung. Aus der Krise Anfang der 90er ist man gestärkt hervorgegangen, indem man das Konzept des Mainframes änderte. Da viele IT-Zentren, hauptsächlich aus dem Finanz- und Versicherungsbereich, auch heute noch auf Mainframes zugreifen, weil sie keine Alternative zur Hochverfügbarkeit und Sicherheit sehen, werden Mainframes auch in Zukunft eine wichtige Rolle spielen.

3 Glossar

AES	**A**dvanced **E**ncryption **S**tandard ist ein symmetrisches Kryptosystem	12
APPN	**A**dvanced **P**eer-to-**P**eer **N**etworking ist eine Erweiterung der Systems Network Architecture-Protokollsuite von IBM aus dem Jahre 1985.	10
Architektur	Das Zusammenspiel der Komponenten eines komplexen Systems.	2, 5, 8, 10, 11, 17
ASCC	**A**utomatic **S**equence **C**ontrolled **C**alculator ist ein in den USA zwischen 1943 und 1944 vollständig aus elektromechanischen Bauteilen gebauter Computer	2
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange ist eine 7-Bit-Zeichenkodierung	6
BCD	BCD oder BCD-Code (von engl. **B**inary **C**oded **D**ecimal) bezeichnet einen numerischen Code, der jede Ziffer einer Dezimalzahl einzeln dualkodiert	6
BOS/360	**B**asic **O**perating **S**ystem/360	7
BPS	**B**asic **P**rogramming **S**upport	7
CF	**C**oupling **F**acility ist eine Hardware-Komponente von IBM-Großrechnern. Sie hat die Aufgabe, allen Prozessoren Zugriff auf die gleichen Daten zu ermöglichen, und ist zudem für die flexible Lastverteilung und die Skalierung zuständig.	11
Client-Server-Architektur	Das Client-Server-Modell beschreibt eine Möglichkeit, Aufgaben und Dienstleistungen innerhalb eines Netzwerkes zu verteilen. Die Aufgaben werden von Programmen erledigt, die in Clients und Server unterteilt werden.	10
CMOS	**C**omplementary **m**etal–**o**xide–**s**emiconductor (dt. komplementärer Metall-Oxid-Halbleiter) ist eine Bezeichnung für Halbleiterbauelemente, bei denen sowohl p-Kanal- als auch n-Kanal-MOSFETs auf einem gemeinsamen Substrat verwendet werden.	11
COBOL	**Co**mmon **B**usiness **O**riented **L**anguage ist eine Programmiersprache, die wie FORTRAN in der Frühzeit der Computerentwicklung 1960 entstand und bis heute verwendet wird	4, 7
CP-67	**C**ontrol **P**rogram stellt als Hypervisorprogramm virtuelle Maschinen für die IBM/360-**67** bereit	7
DAT	**D**ynamic **A**ddress **T**ranslation	8
DB2	relationales Datenbankmanagementsystem von IBM	10, 11
DOS/360	**D**isk **O**perating **S**ystem/360	7

EBCDIC	Der **E**xtended **B**inary **C**oded **D**ecimals **I**nterchange **C**ode ist eine von IBM entwickelte 8-Bit-Zeichenkodierung, bei der jedoch nicht alle Codewörter verwendet werden.	6, 16
ENIAC	Der **E**lectronic **N**umerical **I**ntegrator **a**nd **C**omputer war der erste rein elektronische Universalrechner.	2
ENVAC	**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer ist ein von J. Presper Eckert und John W. Mauchly konstruierter Computer aus den späten 40er Jahren.	2
ESCON	**E**nterprise **S**ystem **Con**nection ist ein Protokoll, das von Mainframes verwendet wird, um den Datenaustausch zwischen dem Rechner und dessen Peripheriegeräten durchzuführen.	11, 12
FORTRAN	**For**mula **Tran**slation ist eine prozedurale und in ihrer neuesten Version zusätzlich eine objektorientierte Programmiersprache, die insbesondere für numerische Berechnungen eingesetzt wird	2, 4, 7
Hypervisor	eine Virtualisierungssoftware, die eine Umgebung für virtuelle Maschinen schafft	9
IBM	**I**nternational **B**usiness **M**achines Corporation (IBM) ist ein US-amerikanisches IT- und Beratungsunternehmen mit Sitz in Armonk bei North Castle im US-Bundesstaat New York.	1, 2, 4, 5, 6, 7, 8, 10, 11, 12, 13, 16, 17
IBM 700/7000 Serie	Reihe von IBM Mainframe-Systemen aus den 1950er und frühen 1960er Jahren	2, 4
IBM Solid Logic Technology	IBMs Methode zur Verkapselung elektronischer Schaltungen eingeführt in 1964 mit der IBM System/360 Serie und verwandten Maschinen	5
Installationen	Einrichtung	4, 8
JES	**J**ob **E**ntry **S**ubsystem ist ein Subsystem unter z/OS auf IBM-Großrechnern	10, 17
Linux in zSeries	Portierung von Linux auf die IBM Großrechner-Plattform System z	12
LPAR	**L**ogical **Par**tition ist die Aufteilung eines IBM-Großrechners (Mainframe) in mehrere virtuelle Systeme.	9
Mainframe	Ein Großrechner (engl.: mainframe) ist ein sehr komplexes und umfangreiches Computersystem, das weit über die Kapazitäten eines Personal Computers und meist auch über die der typischen Serversysteme hinausgeht.	1, 2, 4, 6, 8, 10, 11, 12, 13, 16, 17
MFT	**M**ultiprogramming with **F**ixed number of **T**asks ist Steuerungsprogramm des Betriebssystems OS/360	7, 8, 9
MVS	**M**ultiple **V**irtual **S**torage war das gebräuchlichste Betriebssystem auf den IBM-Großrechnern System/370 und System/390	9, 17

MVT	**M**ultiprogramming with **V**ariable number of **T**asks ist Steuerungsprogramm des Betriebssystems OS/360	7, 8, 9
OS/360	OS/360 war ein Betriebssystem mit Stapelverarbeitungsfähigkeit, das die IBM für ihre System/360-Großrechner im Jahr 1964 angekündigt hatte. OS/360 wurde als eine Familie bestehend aus drei Steuerungsprogrammen (PSP, MFT, MVT) entwickelt.	7
OS/VS1	Betriebssystem mit Virtualisierung	9
OS/VS2 SVS	Erster Release von OS/VS2 mit Unterstützung von einem einzelnen virtuellen Adressraum (engl. **S**ingle **V**irtual **S**torage)	9
PCP	**P**rimary **C**ontrol **P**rogram ist das einfachste Steuerungsprogramm des Betriebssystems OS/360	7
Prozessor	Der Hauptprozessor (engl. central processing unit, CPU), oft auch nur als Prozessor bezeichnet, ist die zentrale Verarbeitungseinheit eines Computers, die in der Lage ist, ein Programm auszuführen.	9, 11, 12
RACF	**R**esource **A**cces **C**ontrol **F**acility ist IBMs Implementierung der Sicherheitschnittstelle SAF (System Authorization Facility) der Großrechnerbetriebssysteme MVS und z/VM.	10, 17
RAM	**R**andom-**A**ccess **M**emory ist ein Halbleiterspeicher	8
RMF	**R**esource **M**easurement **F**acility ist ein Software-Monitor der IBM für das z/OS-Betriebssystem.	9, 17
SNA	**S**ystems **N**etwork **A**rchitecture ist eine Netzwerkarchitektur, die von IBM in den 1970er-Jahren entwickelt und im Jahre 1974 vorgestellt wurde.	10
Sysplex	**Sys**tem processing com**plex** ist eine um 1990 von IBM eingeführte lose Rechnerkopplung von IBM Großrechnern.	11, 12, 17
System z	System z (früher zSeries) ist die im Jahr 2000 eingeführte, aktuelle Großrechnerarchitektur der Firma IBM.	4, 12, 17
System z196	zEnterprise 196 (z196) ist die neueste (22. Juli 2010) Generation von Mainframes mit bis zu 96 Prozessoren.	13
TCPIP	**T**ransmission **C**ontrol **P**rotocol / **I**nternet **P**rotocol ist eine Familie von Netzwerkprotokollen und wird wegen ihrer großen Bedeutung für das Internet auch als Internetprotokollfamilie bezeichnet.	10
TOS/360	**T**ype **O**perating **S**ystem/**360** von IBM	7
TSO	**T**ime-**S**haring **O**ption ist ein interaktiver Kommandozeileninterpreter für IBM-Großrechner-Betriebssysteme z/OS.	9, 17
TSS/360	**T**ime**s**haring **S**ystem	7

UNICODE	Unicode ist ein internationaler Standard, in dem langfristig für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird.	6
UNIVAC I	**Univ**ersal **A**utomatic **C**omputer I war der erste kommerziell hergestellte Computer in den USA.	2
VM	**V**irtuelle **M**aschine	7, 8, 9, 11, 17
VM/370	Trägersystem für andere Betriebssysteme	7
VTAM	**V**irtual **T**elecommunications **A**ccess **M**ethod stellt u.a. eine Programmierschnittstelle (API) bereit, die Funktionen zur Kommunikation mit Geräten für die Datenfernübertragung und deren Benutzern implementiert.	10
Watson, Thomas J., Jr.	Chef von IBM von 1952 bis 1971	2
z/VM	Betriebssystem für (fast) beliebige Anzahl von **v**irtuellen **M**aschinen (VM) auf einem Großrechnersystem der **z**Serie von IBM	7, 12
z/VSE	**V**irtual **S**torage **E**xtended on **z**/Architecture ist ein Betriebssystem für IBM-Mainframe-Computer	7
Zero Downtime	Das **z** im Namen des Mainframes steht für *zero downtime*.	12

4 Literaturverweise

1. The History of the Mainframe Computer. www.vikingwaters.com/htmlpages/MFHistory.htm, 2010.
2. IBM Archives, IBM 701. www-03.ibm.com/ibm/history/exhibits/701/701_intro.html, 2010.
3. IBM Archives Mainframes product profiles, 704 Data Processing System. www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP704.html, 2010.
4. Wikipedia Metrovick 950. en.wikipedia.org/wiki/Metrovick_950, 2010.
5. IBM Archives Mainframes product profiles, 7090 Data Processing System. www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7090.html, 2010.
6. IBM Archives Mainframes basic information, System/360 Announcement. www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PR360.html, 2010.
7. IT-Wissen EBCDIC. www.itwissen.info/definition/lexikon/extended-binary-coded-decimal-interchange-code-EBCDIC-EBCDIC-Code.html, 2010.
8. Wikipedia EBCDIC. en.wikipedia.org/wiki/Extended_Binary_Coded_Decimal_Interchange_Code, 2010.

9. IBM Puplib EBCDIC. pub-lib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zappldev/zappldev_14.htm, 2010.
10. History of IBM mainframe operating systems, System/360 operating systems. en.wikipedia.org/wiki/History_of_IBM_mainframe_operating_systems#System.2F360_operating_systems, 2010.
11. Computerwoche, Zum 40. Geburtstag des S/360-Großrechners. www.computerwoche.de/nachrichtenarchiv/545687/index.html, 2010.
12. Wikipedia System/360, Namensbedeutung. de.wikipedia.org/wiki/System/360#Namensbedeutung, 2010.
13. IBM Entwicklung 1970. www-05.ibm.com/de/entwicklung/history/1970.html, 2010.
14. Wikipedia VM. en.wikipedia.org/wiki/VM_(operating_system), 2010.
15. Computerwoche, „IBM feiert den Mainframe-Mythos“. www.computerwoche.de/nachrichtenarchiv/545687/index.html, 2010.
16. Wikipedia TSO. de.wikipedia.org/wiki/Time-Sharing_Option, 2010.
17. Wikipedia MVS. en.wikipedia.org/wiki/MVS, 2010.
18. IBM zOS Features – RMF. www-03.ibm.com/systems/z/os/zos/features/rmf/, 2010.
19. IBM Publib RACF. pub-lib.boulder.ibm.com/infocenter/zos/basics/index.jsp?resultof=%22racf%22%20, 2010.
20. IBM Publib JES. pub-lib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zconcepts/zconc_whatisjes.htm, 2010.
21. Wikipedia System Network Architecture. de.wikipedia.org/wiki/Systems_Network_Architecture, 2010.
22. IBM Pubib “Dynamic Channel Path Management”. pub-lib.boulder.ibm.com/infocenter/zos/v1r9/index.jsp?topic=/com.ibm.zos.r9.ieaw100/iea2w180109.htm, 2010.
23. IBM System/370 Announcement. www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PR370.html, 2010.
24. Wikipedia System/390. en.wikipedia.org/wiki/IBM_ESA/390, 2010.
25. IBM System/390 Announcement. www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PR390.html, 2010.
26. Wikipedia Coupling Facility. de.wikipedia.org/wiki/Coupling_Facility, 2010.
27. Wikipedia IBM Parallel Sysplex. en.wikipedia.org/wiki/IBM_Parallel_Sysplex, 2010.
28. Wikipedia E-Business. de.wikipedia.org/wiki/E-Business, 2010.
29. Wikipedia System z. de.wikipedia.org/wiki/System_z, 2010.
30. Abbildung 4, eServer zSeries 990. www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_2423PH990.html, 2010.
31. Wikipedia System z. en.wikipedia.org/wiki/IBM_System_z, 2010.

Kaiserslautern Mainframe Summit 2010: Hardware of an IBM Mainframe zEnterprise 196

Arvind Korandla, Dimitri Blatner, Maximilian Senftleben
arvindreddy11@gmail.com, d_blatne@cs.uni-kl.de,
m_senftl@cs.uni-kl.de

Database and Information Systems Group
University of Kaiserslautern
Supervisor: Karsten Schmidt

Abstract. This paper will summarize the specifications of the hardware and peripheries including their layout used in today's Mainframes from IBM and describe the new technologies and improvements in comparison to older IBM Mainframes in view of hardware and peripheries. Some further in-depth information about special working techniques will be provides by other papers of this seminar.

1 Introduction

Due to the fact, that Mainframes are still used and important for big companies today to handle lots of critical transactions/queries and that applications nowadays become more and more service-oriented and complex, the requirements for hardware also raise very fast. To meet these requirements for hardware components and architecture many features and improvements were achieved. And everybody, who wants to work with Mainframes at a system programming level or even hardware engineering level should know the different components in depth.

We try to give an overview of the most important parts of a Mainframe. This paper is divided in three parts, first of all the *Processors & Memory*, followed by *Layout & Configurations*, *Data Center Implementation*, *Peripheral Devices* and finally dealing with *Interconnections*. Since, there are tons of abbreviations you should sometimes take a look in our glossary. You can see an overview in **Figure 1**.

1.1 Overview

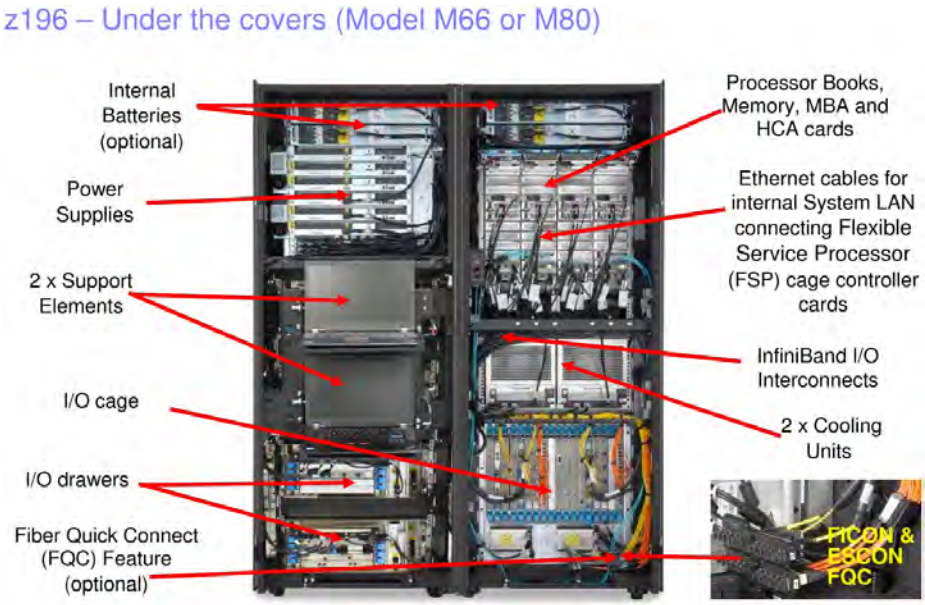


Fig. 1. Current IBM zEnterprise Mainframe z196 under the cover. You can see the systematic layout of the different parts. [1]

2 Processors & Memory

In the following subsections the computational heart of the z196 mainframes will be explained.

2.1 Books

In **Figure 2-1** you can see the internal layout and number of components of a Book. Each Book contains a Multi-Chip-Module (MCM) consisting of multiple processors, main memory, I/O connectors, a cooling and power system. Each component will be described in more detail later.

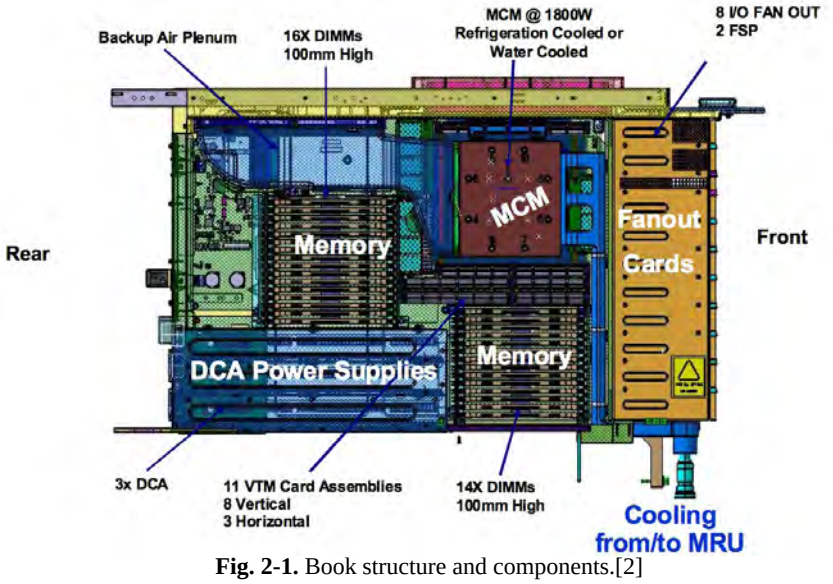


Fig. 2-1. Book structure and components.[2]

The most important advantages of the book architecture are to have all components belonging to each other directly in one shell – everything in one box - and to be able to plug or unplug them easily.

In terms of high-availability of mainframes Books can be (un-)plugged while the system is running. Also nearly every part inside a Book (and inside a mainframe) is redundant.

The Books are interconnected via fiber channel cables in point-to-point topology as you can see in **Figure 2-2**:

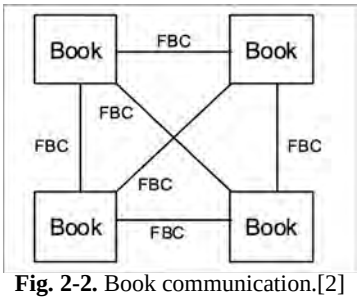


Fig. 2-2. Book communication.[2]

2.2 Processor Units - PUs

The z196 PUs are completely new developed full-custom multicore CMOS chips produced with SOI technology and a minimal gate length of 45 nm for a transistor. It is promoted as a high tech CISC processor, including a very large instruction-set and modern execution techniques like superscalarity, dynamic scheduling, out-of-order execution, but more details in the section *2.2.1 Micro-Architecture And Instruction Set*.

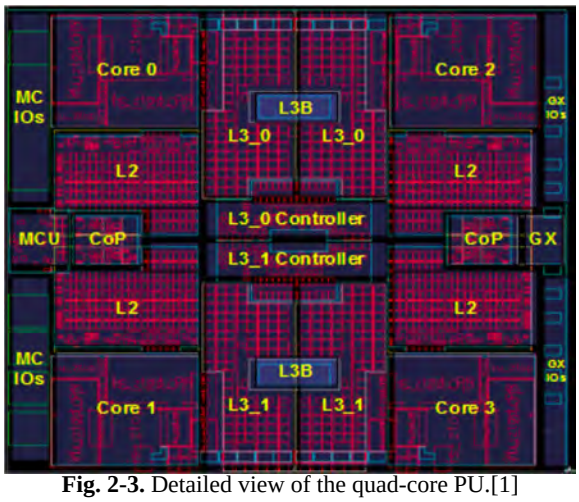


Fig. 2-3. Detailed view of the quad-core PU.[1]

Only Intel can produce smaller transistors (32 nm) at the moment. Look at **Figure 2-3** and **Figure 2-4** for detailed design of the PU and memory. Here are some other facts:

- fully 64 bit
- 4 cores per PU
- 2 co-processors (CoP)
- 5.2 GHz frequency
- Cache
    - Level 1: 64 KB instruction per core, 128 KB data per core
    - Level 2: 1.5 MB per core
    - Level 3: 24 MB shared over cores
    - Level 4: 192 MB shared over SCs (see section 2.3)
- 23.5 mm × 21.8 mm = 512.3 mm^2 chip area
- 13 layers of metal
- 1.4 billion transistors
- 3.5 km of wire

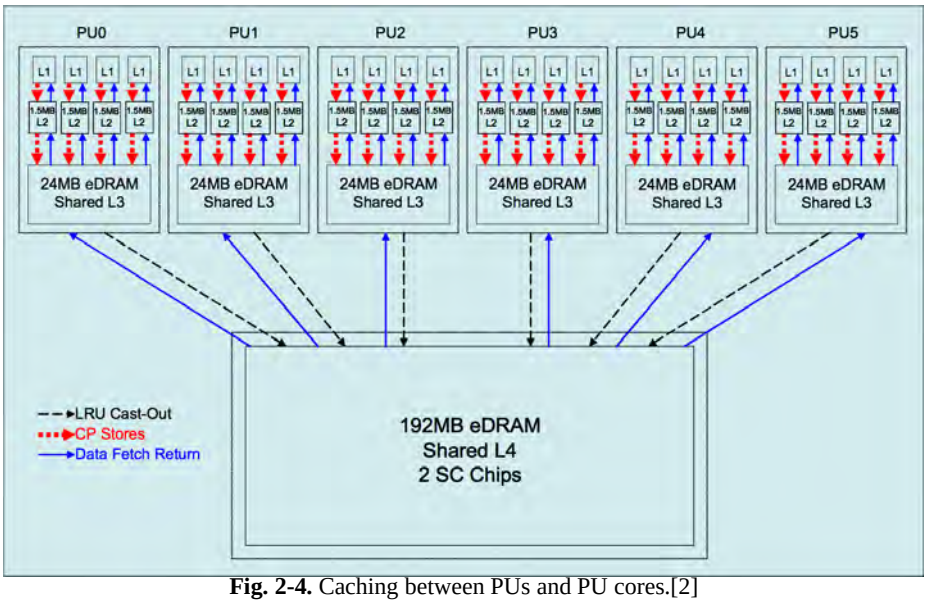


Fig. 2-4. Caching between PUs and PU cores.[2]

2.2.1 Micro-Architecture And Instruction-Set

There are two different types of processors, RISC and CISC. In comparison to CISC, RISC processors have a reduced instruction set and a much simpler circuit complexity. Almost every general purpose processor has an internal RISC structure, because of a much easier implementation of pipelining and a much more compact micro-architecture of the computational logic. The PUs in IBM's mainframes are promoted as CISC processors, not only in terms of the large instruction set. They have 762 fully in hardware implemented functions and a total of 984 instructions. Further they support multiple addressing modes (24, 31 and 64 Bit) to support applications programmed in the 1960s and multiple arithmetic formats, e.g. a saturation arithmetic. So it can be said, that the PUs are more CISC processors, though the execution units of each processor core have RISC design.[1]

The architecture supports high-grade virtualization with multiple address spaces, to run up to 1000's of operating systems fully isolated from each other.

But why are there so many different instructions? One point is the pricing model of IBM, where basically the number of cycles a PU works matters. Another point is that IBM wants to support even the oldest Mainframe applications, which maybe work on an old or different instruction set. In fact there are companies, that still use applications programmed before 1970 and they are working fine. The third point is that applications running with special instructions need less PU cycles and are more efficient or faster.

The newest z196 system also provides new special instructions for cryptography, since security is and becomes a more and more important topic. Therefore the PUs can en-/decode encrypted data very fast or even work directly with encrypted data, instead of decoding it and writing the encoded data into memory. Compared to the z10 top model, the z196 can achieve up to 30% more performance with recompiling the same application.

Superscalarity means, that one PU can fetch multiple instructions per clock cycle from the memory or cache.

Out-Of-Order execution is closely related to superscalarity. The instructions will be analyzed for data dependencies and sorted topologically in a sort of instruction table. This has the advantage, that if more instructions are independent, they can be executed in parallel.

Each PU core has six execution units: two for fixed point arithmetic, two for load/store operations, one for binary floating point arithmetic and one for decimal floating point arithmetic.[1]

2.3 Storage Control - SC

Like the PUs the storage control (SC) chips use 45nm CMOS SOI technology, again 13 metal layers. With 24.4 mm × 19.6 mm = 478,24mm² a SC is less sized and with 1.5 billion it has more transistors. The largest part is the eDRAM with 1 billion transistors for 96 MB of data, as you can see in **Figure 2-5**. The storage of the SC serves as a shared Level-4 cache for all the PUs on the MCM and manages the clock ratio for the communication with Level-3 caches on each PU. Every MCM has two SCs – implying 192 MB cache per MCM/Book - communicating with each other and with SCs from up to three other Books.

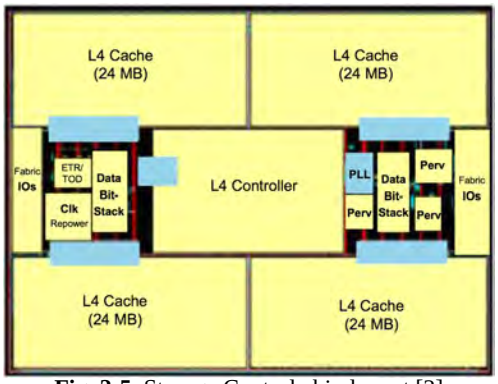


Fig. 2-5. Storage Control chip layout.[2]

2.4 Memory

Each Book can have up to 960 GB of physical memory and up to 768 GB usable memory. The remaining memory is used for redundancy. IBM calls it the *Redundant Array of Independent Memory* (RAIM). There are 30 available memory slot per Book, each slot equipped with a 4 GB, a 16 GB or a 32 GB DIMM.

The slots are arranged in five channels, every PU is connected through the Memory Control Units (MCU) to all of the five channels. One channel is always used for parity. **Figure 2-6** and **Figure 2-7** show it in more detail:

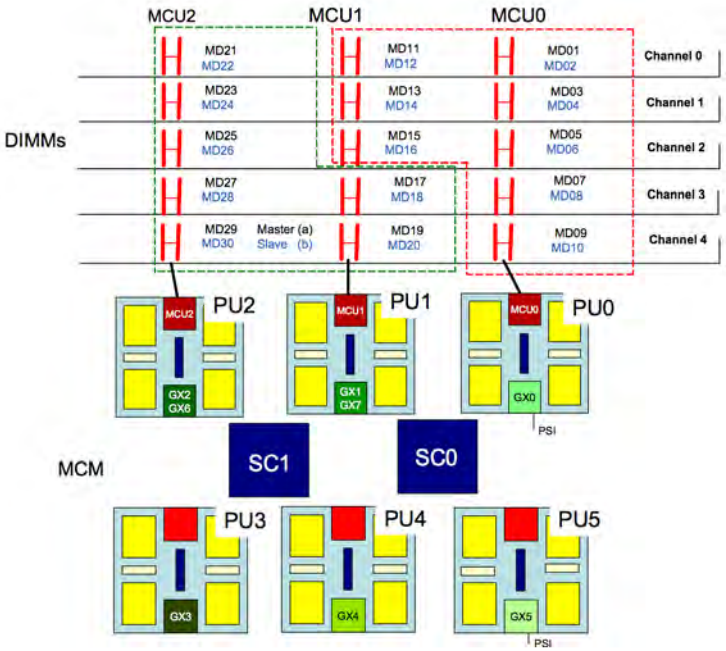


Fig. 2-6. Memory to MCM interconnection.[2]

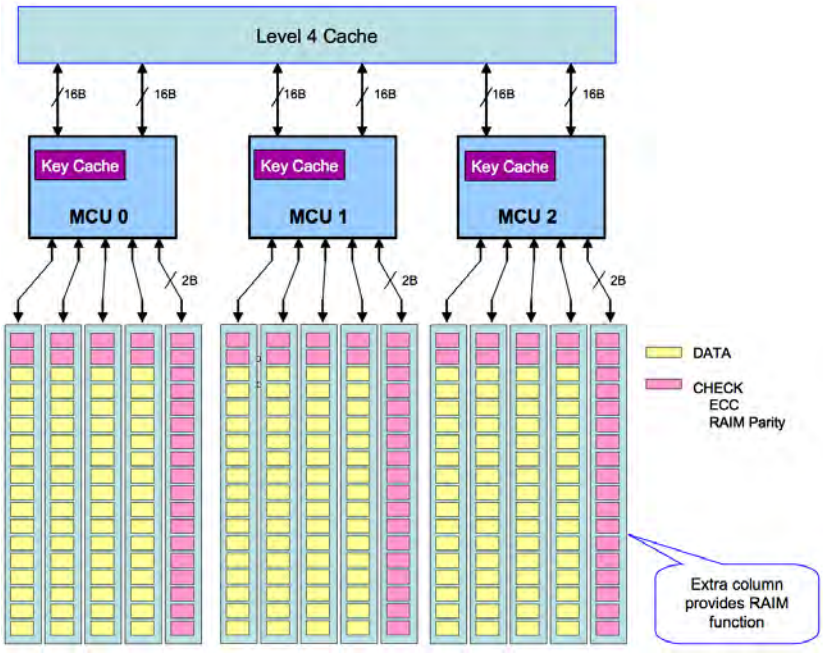


Fig. 2-7. Parity DIMMs in RAIM.[2]

The RAIM design is able to detect every DRAM, socket, memory channel oder DIMM failures and to correct them during runtime, look at **Figure 2-7**.

2.5 MCMs

The heart of every Book is a Multi-Chip-Module (MCM), that is a 96 mm × 96 mm = 9.216 mm^2 sized plate, made by glass ceramic with 103 layers.

It contains six PUs and two SCs on it. That makes a total of 24 PU cores working with 192 MB shared Level-4 cache.

Again, due to high-availability reasons, two PUs are redundant. Hence 16 cores per Book can be used simultaneously. In the highest version of the z196 mainframe there are five Books with a total of 80 usable processor cores, the leftover cores were used for redundancy.

In **Figure 2-8** the *S##* marked chips contain product data information about the MCM and other engineering information. Only two of them are active, the other two are – again – redundant.

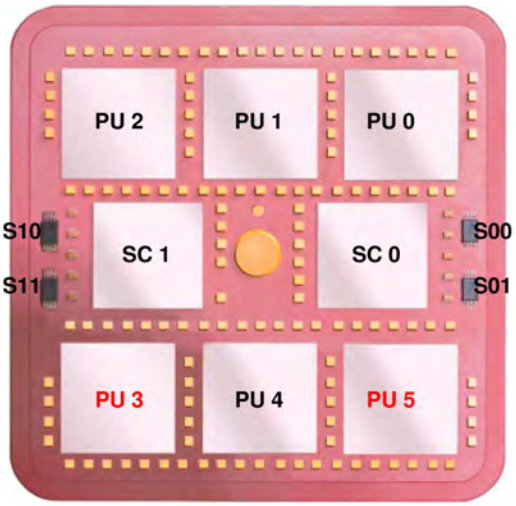


Fig. 2-8. Layout of a MCM.[1]

2.6 HCA/MBA Fanout Cards

Every Book has fanout slots for up to eight so called Host Channel Adapter (HCA) – Memory Bus Adapter (MBA) in older z9 mainframes – fanout cards as you can see in **Figure 2-1**. These cards are only used for I/O operations and typically connected to the I/O drawers or I/O cages of the mainframe (more details later).

There are three different types of HCA:

- HCA2-C: Copper fanout connected to IFB-MP cards in an I/O cage or drawer
- HCA2-O: Optical fanout with 12 coupling links (InfiniBand protocol), distance up to 150 meters, bidirectional transfer

- HCA2-O LR: Optical fanout with 1 coupling link (InfiniBand), distance up to 10 km, unidirectional transfer

Considering the usage of eight HCA cards, one Book can achieve a speed of 8 × 6 Gbps = 48 GBps, bidirectional.

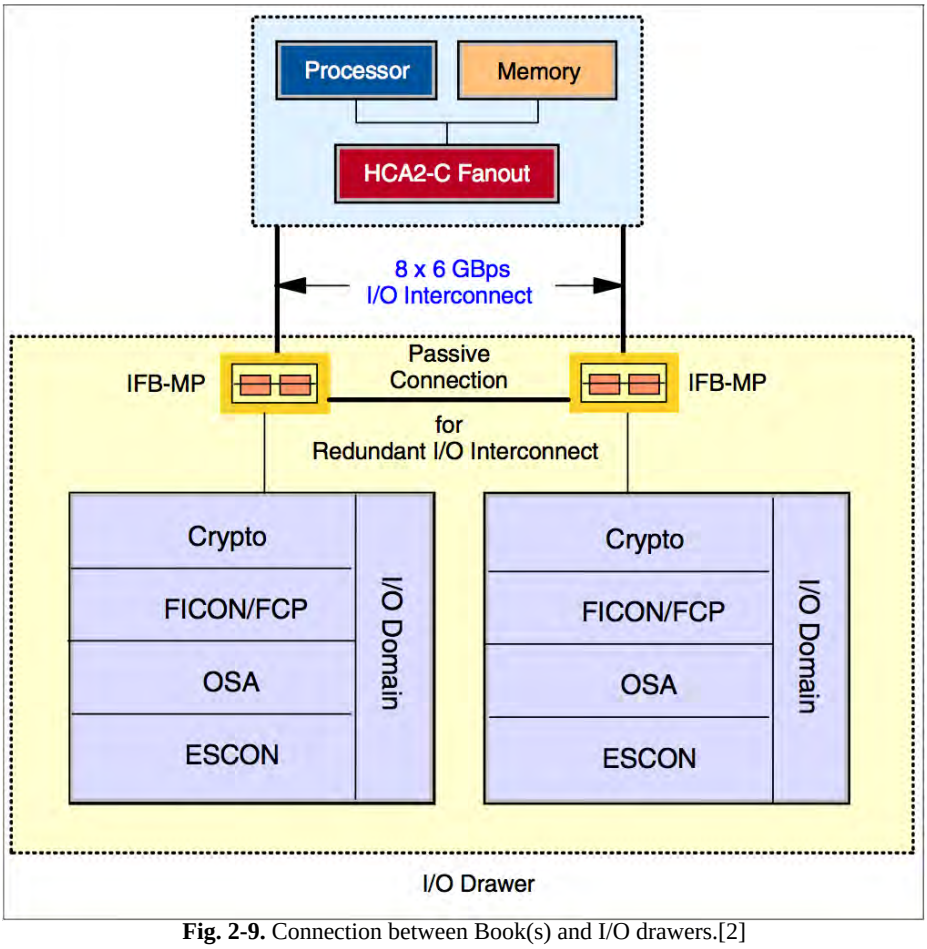


Fig. 2-9. Connection between Book(s) and I/O drawers.[2]

2.7 Special PU Versions & Pricing

There are some *"versions"* of PUs that have been introduced, all based on the same processor architecture, but limited respectively specialized for certain tasks:

- CP: Central Processor – this is the "full" processor, that supports z/OS; customers pay per PU cycle
- SAP: System Assistance Processor – not visible for applications as a processor, it executes microcode for the I/O subsystem (e.g. converting

addresses) of the mainframe and disburdens the application PUs; at least one PU in a mainframe is a SAP and more in larger mainframes

- IFL: Integrated Facility for Linux – specialized PU for running Linux and/or z/VM, z/OS cannot be used; payed once per IFL
- zAAP: System z Application Assist Processor – specialized PU for running Java and XML instructions; very limited in other functionality, so no operating system can be run on a zAAP; payed once per zAAP
- zIIP: System z Integrated Information Processor – only used for certain workloads, to support regular PUs and reduce their cycles (e.g. recurring tasks in databases); payed once per zIIP
- ICF: Integrated Coupling Facility – used for management of Sysplex coupling of mainframes (or Parallel Sysplex and Geographical Dispersed Parallel Sysplex);

The really new thing introduced with the z196 mainframe is the IFL, to make running Linux on a mainframe attractive. In general these *special* PUs are nothing but full PUs with limited functionality in different parts and help reducing the software costs dramatically, since full CPs are accounted by cycle numbers.

The PUs inside a mainframe can be configured at any point of time, to be a different kind of PU (e.g. an IFL), without the need to change the hardware.[1]

3 Layout & Configurations

The z196 consists of two frames bolt together, as shown in **Figure 3-1**, known as "A frame" and "Z frame", both build to meet the 42U Electronic Industry Association (EIA) frame standard.

There are several possibilities of customization based on customer requirements. z196 supports up to four I/O drawers, three I/O cages (a third I/O cage requires ordering of RPQ 8P2506) or combinations of both to provide IO interfaces for the processing books. [2]

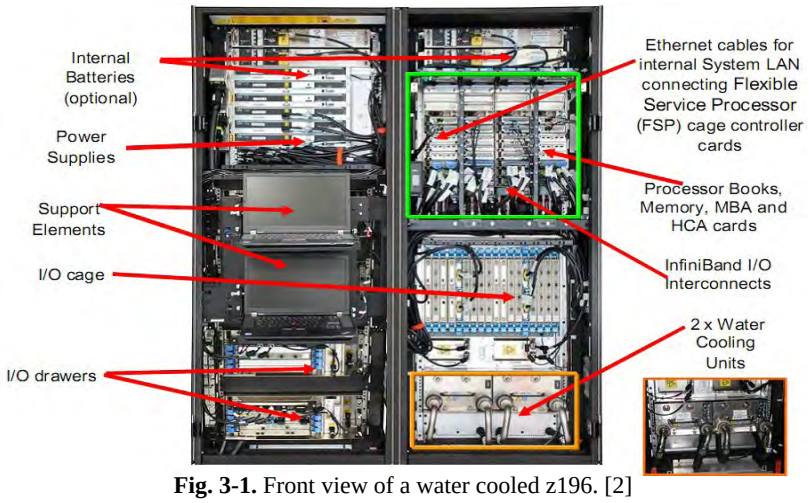


Fig. 3-1. Front view of a water cooled z196. [2]

3.1 “A Frame”

The "A frame" contains the Central Processing Complex (CPC) cage at the top which itself consists of up to four processing books with Multi-Chip Module (MCM) and memory. Optionally two Internal Battery Feature (IBF) can be placed above the CPC.

Beneath the CPC the modular cooling units, either two modular refrigeration units (MRU) or two water cooling units (WCU), are placed as well as one of the following configurations: [2]

- No I/O cage / drawer
- One or two I/O drawer
- One I/O cage

3.2 “Z Frame”

The "Z frame" contains the Bulk Power Assemblies (BPA) and optionally two times two Internal Battery Feature (IBF) at the top.

Beneath the BPA one of the following configurations is placed:

- No I/O cage / drawer
- Up to four I/O drawer
- One or two I/O cages (second I/O cage requires ordering of RPQ 8P2506)
- Combinations of up to four I/O drawer and up to two I/O cages

In front of I/O cage 2 the Support Element tray is located containing two Support Elements (SE). The z196 is shipped with a pair of Thinkpads as SEs. A SE can be connected to one or more Hardware Managment Consoles via Ethernet/LAN for monitoring and controlling (e.g. POR: Power-On Reset). [2]

3.3 I/O drawers, cages

Each of the processing books in the CPC has eight dual-port fanout cards for data transfer with 6 GBps bidirectional bandwidth per port. Altogether the 16 IFB (InFiniBand) I/O interconnects aggregate a bandwidth of 96 GBps per book.

Each domain of an I/O drawer/cage is connected by an IFB-MP (InFiniBand Multiplexer) card via copper cable to the Host Channel Adapter (HCA) of the CPC supporting a link rate of 6 GBps. Power is distributed in the I/O drawer/cage by two Distributed Converter Assemblies (DCA). [2]

The number of required I/O drawers and cages are determined by the number of I/O feature cards as shown in **Table 3-1**.

Table 3-1. Number of I/0 drawers / cages based on number of I/O features [2]

# I/O feature cards	# I/O drawers	# I/O cages
1-8	1	0
9-16	2	0
17-24	3	0
25-32	4	0
33-36	1	1
37-44	2	1
45-52	3	1
53-60	4	1
45-56	0	2
57-64	1	2
65-72	2	2
73-84 (RPQ req.)	0	3

3.3.1 I/O drawer

An I/O drawer supports up to 8 horizontally installed I/O slots in two I/O domains. In contrast to I/O cages drawers can be added or removed non-disruptively. [3]

Figure 3-2 shows a sketch of an I/O drawer. I/O drawer are installed vertically.

3.3.2 I/O cage

An I/O cage supports up to 24 vertically installed I/O slots separated in seven I/O domains, as you can see in **Figure 3-2**. [3]

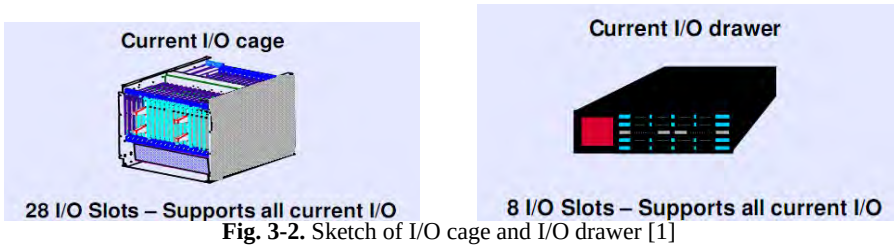


Fig. 3-2. Sketch of I/O cage and I/O drawer [1]

4 Data Center Implementation

The z196 is installed on a raised floor, therefore weight and size need to be paid special attention to.

- Size
  - A z196 consisting of A frame and Z frame has a width of 1,534 m, a height of 2,012 m and a depth of 1,273 m for an air-cooled server or 1,375 m for a water-cooled server. [2]

- Weight
  - The weight of a z196 (A and Z frame) depends mainly on the number of I/O units, the chosen cooling method and the installation of IBFs. It ranges from 1894 kg with one I/O frame, air-cooling and no IBFs up to 2185 kg with three I/O cages, water-cooling and IBFs installed.
  - IBM offers different methods to handle the systems weight. One method is an weight distribution plate that distributes the weight over more floor panels. Another is the 3-in-1 bolt down kit to stabilize the frames and tie them down to a concrete floor beneath the raised floor. [2]

- Power
  - The z196 has two independent redundant power supplies. Each linked individual either one-fold or double depending on customer configuration to two different power feeds.
  - Line cords attach to either 3 phase, 50/60 Hz, 200 to 480 V AC power or 380 to 520 V DC power.
  - Power consumption depends on cooling method, number of processing books and number of I/O units.
    Maximum Power requirement for small system z196 M15 with zero I/O Units is up to 6.8 kVA and for big system z196 M80 with 6 I/O Units up to 30.1 kVA. [2]

- Internal Batterie Feature (IBF)
  - With an IBF it is possible to keep system operations up for a short time if both power feeds are disrupted.
  - The estimated IBF hold-up times depend on power consumption respectively the z196 hardware configuration and can be seen in **Table 4-1**. [2]

Table 4-1. Estimated battery hold-up times in minutes. [2]

	# I/O Units						
	0	1	2	3	4	5	6
M15	6	5	4	9	7	6.8	6.8
M32	7.5	7	6	9.5	7.5	7	6
M49	8	7.4	7	6	5	4.6	4.1
M66 /M80	5	4.8	4.5	4	3.6	3.2	3

- Cooling
  Cooling is essential for mainframe operation because the hardware has certificated operation temperature ranges.
  Cooling is achieved either by refrigeration assisted air-cooling or optionally by chilled water cooling. This configuration is factory installed and can not be changed afterwards. [2]

- Air cooling
  1. The air cooling is realized a a hybrid cooling system consisting of Modular Refrigeration Units (MRU), Motor Scroll Assemblies (MSA), Motor Drive Assemblies (MDA) and Bulk Power Regulator (BPR).
  2. The BPR regulates cooling by controlling the blower speed. The one to two MRUs provide a closed-loop liquid cooling subsystem to refrigerate the content of the processing books.
  3. If one of the MRUs fails, backup blowers compensate it with additional air cooling. If the books get to hot, the cycle time is reduced dynamically by up to 17% of the maximum capacity by the oscillator card. [2]
- Water cooling
  1. Water cooling is introduced by the z196. The water cooling unit (WCU) allows cooling with customer chilled water.
  2. The water cooling has different water supply requirements for flow rate and pressure based on the incoming water temperature and installed hardware configuration.
  3. A water cooled model comes along with rear-door heat exchangers for each frame to optimize refrigeration and two independent WCUs for availability reasons. [2]

- Cabling
  - Cables can be installed in the raised floor as well as using the optional top exit introduced with the z196 on top of the frames. The optional top exists support the fiber optical cables (ESCON, FICON, OSA, InfiniBand, ISC-3) as well as the copper cables for Ethernet features. [2]

5 Peripheral devices

The System z consists of different peripheral devices to form a environment for productive usage.

5.1 Storage

Storage conceptually is still an arrangement of IBM 3990 Control Units and IBM 3390 Disc Units for compatibility reasons. Nowadays products like the IBM TotalStorage DS8000 emulate large numbers of control and disc units while offering new functionality and better reliability.

- Disk storage
  - The physical HDDs are common SCSI-type units operating for example in many RAID5 arrays.
  - IBM 3390 disks are available with capacities of 1, 2, 3 and 9 GB.
  - The DS8000 offers functions not native to 3390 devices but frequently used in modern mainframes: FlashCopy, Extended

Remote Copy, Concurrent Copy, ParallelAccess Volumes, Multiple Allegiance. [2][4]

- Tape storage
    - Tape storage is a sequential data access method. It is used to archive data not used in production any longer or accessed less frequently. IBM offers cartridges with capacities of 100, 500 and 700 GB for usage with IBM TS1120 Tape drive. [2]

5.2 Hardware Management Console (HMC)

A HMC manages one or more IBM System z servers. It offers functions for management, monitoring and operating. A HMC is connected to a server through its Support Elements and sends commands to them to perform a task. [2][4]

5.3 BladeCenter Extension (zBX)

Beginning with z196 IBM offers the possibility to host Server Blades with similar qualities of service to those of System z and directly connected to the IBM z196 mainframe. The zBX supports up to 112 Blades in 8 BladeCenter chassis and is equipped with redundant components for a higher availability. [2]

6 Interconnections

System z supports different interconnection methods for external and interal connectivity. The following sections will introduce ESCON, FICON and HiperSockets.

6.1 ESCON

In the early 1960's IBM introduced the IBM System/360 which incorporated a Channel I/O subsystem using a parallel I/O interface [5]. This was also called Bus-and–tag parallel channel architecture [6]. This became as a de-facto industry standard. The technology operated at a rate of (4.5 Mbyte/sec) and was a copper based technology.

This technology was succeeded by ESCON (Enterprise System Connection) in the late 1980's. This interface was introduced to overcome the distance, bandwidth, and cable bulk deficiencies associated with the parallel I/O interface [7]. ESCON is basically a data connection which is commonly used to connect the mainframe computers to the peripheral devices like disk storage and tape drives [8]. ESCON is an optical fiber, half –duplex, serial interface. The ESCON architecture is equivalent to Layers 1 and 2 of the Open Systems Interconnection (OSI) reference model published by the International Organization for Standardization [7].

6.1.1 ESCON: Purpose and Implementation

ESCON PCI Adapter was first announced for the Netfinity Server [9]. This enabled the IBM's Netfinity Server family to have direct ESCON attachment to the S/360 Enterprise Servers. This enabled Netfinity intermediate server to act as SNA or multiprotocol gateway for the browser-based clients, and also allow these clients to access the wealth of data that is stored on the enterprise servers without any costly communication software residing on the clients [9].

The ESCON adapter enhances the reliability and performance between applications on the server and the mainframe by allowing direct communication between them. An illustration of this architecture, has been provided in **Figure 6-1**.

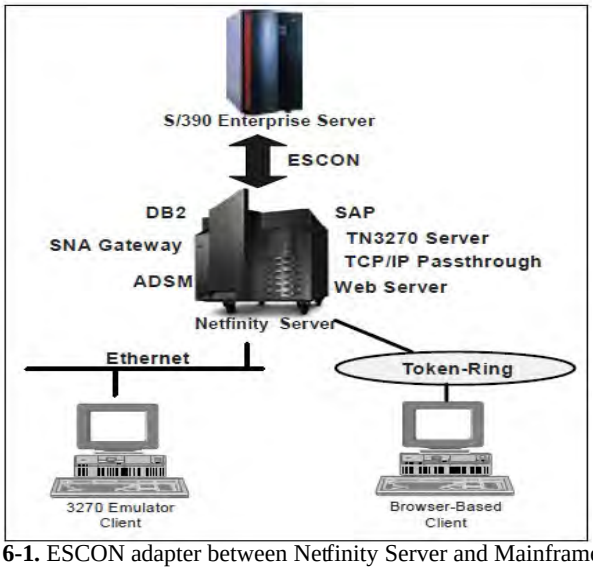


Fig. 6-1. ESCON adapter between Netfinity Server and Mainframe [9]

ESCON channels can be directly connected to the device in a point-to-point configuration, or through an ESCON Director. An ESCON Director allows multiple controllers to connect to one channel or a single controller to connect to multiple channels. The connection can remain in effect while several frames are exchanged between the ports. The switching function provides a way of connecting multiple devices and channels without requiring permanent connections.

Using ESCON's Multiple Image Facility (EMIF) a single controller can connect to multiple Logical Partitions (LPAR's) or the system images. This is depicted in **Figure 6-2**.

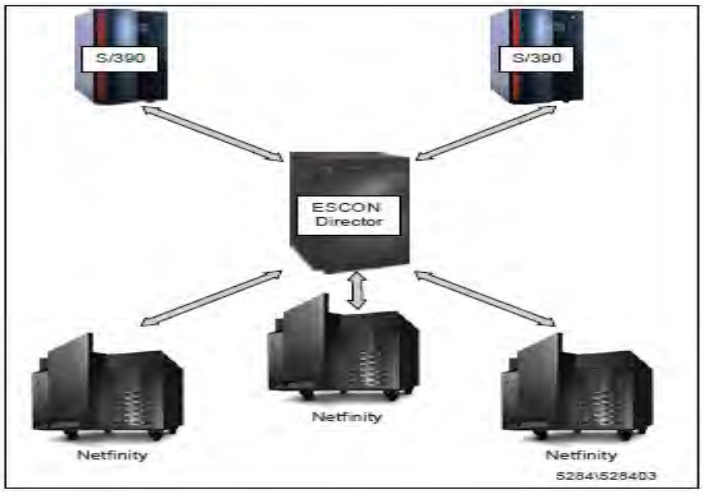


Fig. 6-2. ESCON Director connected to multiple Logical Partitions. [9]

6.2 Disadvantages of FICON over ESCON

The transmission in ESCON is encoded via 8b/10b signaling [7]. The ESCON signaling rate is 200 megabits per second (Mbps), which provides a maximum of 20 megabits per second (Mbps) link-level throughput at distances up to 8km. This approximates to 17Mbps throughput. The throughput decreases as distance increases beyond the 8km mark. ESCON is rather quickly losing its market share to FICON, which is being strongly supported by IBM [8].

6.3 FICON

Before going through the details of FICON, let us first discuss some of the drawbacks of the ESCON channels and its improvements in FICON.

6.3.1 Drawbacks of ESCON and rise of FICON

First, the 9672 ESCON channel implementation limits the device support to 1024 devices (sub channel/device numbers) per channel, whereas a 9672 FICON channel (in FCV mode) can be used to support 16k devices. This is a big improvement in FICON channels [10].

Second, as discussed above, the performance of ESCON channels decreases drastically beyond a distance of 8km.This point of rapid decrease is called as distance data rate droop point. For FICON channels, links between where a data rate droop point occurs is close to 100km. This allows customers to get the benefit of extended distance from a FICON channel [10].

Third, multiple ESCON channel fibers between two sites can be very expensive. With the use of FICON channels, one would observe 8-to-1 reduction in the number of fibers required to connect to the two sites. This will largely reduce the cost associated with it. [7]

Finally, the dark fiber distance (no retransmission of the fiber signal) in FICON is 10km where as in ESCON it can only be 3km.

6.3.2 FICON: Modes

All the aspects of FICON infrastructure are based on ANSI FC standards. FICON generally operates in two modes: bridged (also known as FCV mode) and native (also known as FC mode) [6].

In the bridged mode, the hosts use the FICON adapters to connect to the ESCON directors. Here a FICON bridge adapter is installed in an ESCON director to facilitate communication. This FICON bridge adapter time-division multiplexes up to eight ESCON signals onto a single FICON channel. All the early ESCON directors do not support the FICON bridge adapter.

In the native mode, FICON uses a modified version of the SBCCS (Single Byte Command Code Set) and replaces the ESCON transmission facilities with the fiber channel transmission facilities [6].

Unlike ESCON, FICON native mode operates in full-duplex mode. Also unlike ESCON, fiber channel switches employ packet switching to create connections between hosts and CU's. The FICON native mode thus takes advantage of the packet switching nature of the Fiber channels to allow close to 32 simultaneously active logical connections per physical channel adapter [6].

6.3.3 Migration from ESCON to FICON

Now let us look at a small example of migration from ESCON to FICON channels. Let us analyze the comparison in S/390 processor in **Figure 6-3**.

Without FICON, an S/390 processor can accommodate up to 256 ESCON channels. With the implementation of FICON, an S/390 processor can accommodate up to the equivalent connectivity of 360 ESCON channels. This number is achieved with 168 ESCON channels and the 24 FICON (in FCV mode) providing the connectivity equivalence of 192 ESCON channels. Thus 168 with 192 ESCON channels equate to 360 [10]

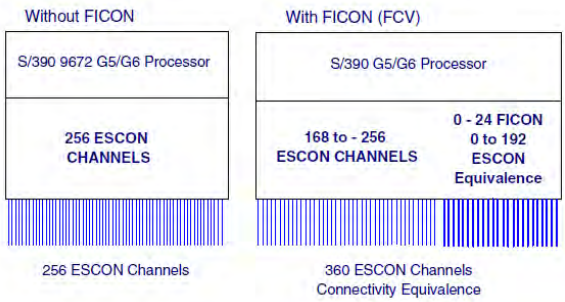


Fig. 6-3. Migration ESCON to FICON. [10]

6.3.4 ESON to FICON (FCV) Aggregation

Figure 6-4 illustrates the result of using FICON channel to consolidate, or aggregate, a number of ESCON channel connections. This configuration is called FICON configuration in FCV mode and uses a FICON bridge port in the ESCON Directory.

The figure on the left of the diagram in **Figure 6-4** shows a number of ESCON (CNC) channels supporting paths to different control units, connected through an ESCON Director. On the right, with the installation of one FICON channel (in FCV mode) in the processor and one FICON Bridge card, the aggregation of eight ESCON channel connections into one FICON channel is possible. This provides additional benefits, such as greater bandwidth, greater distances, and the ability to support up to 8 concurrent operations [10].

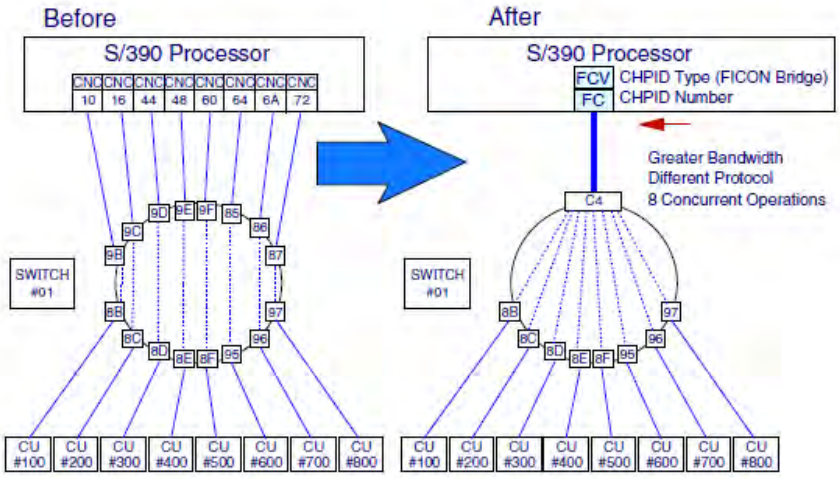


Fig. 6-4. ESCON to FICON bridge aggregation. [10]

6.3.5 Advances in FICON Connectivity

As the time passed by, the performance of FICON increased drastically [6]. The new FICON Express 8 is designed to satisfy the performance requirements of bandwidth hungry applications. FICON Express 8 on z196 and z10, exploiting high performance FICON for System z, is designed to deliver up to 770 Megabytes per second of throughput for full-duplex data transfers [11]. Let us look at the improvements which took place over a period of time in **Figure 6-5** and **Figure 6-6**

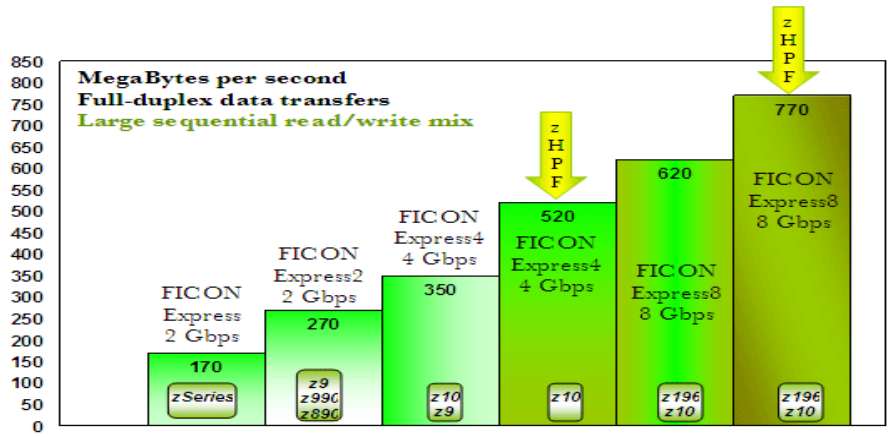


Fig. 6-5. Performance Estimation of FICON. [11]

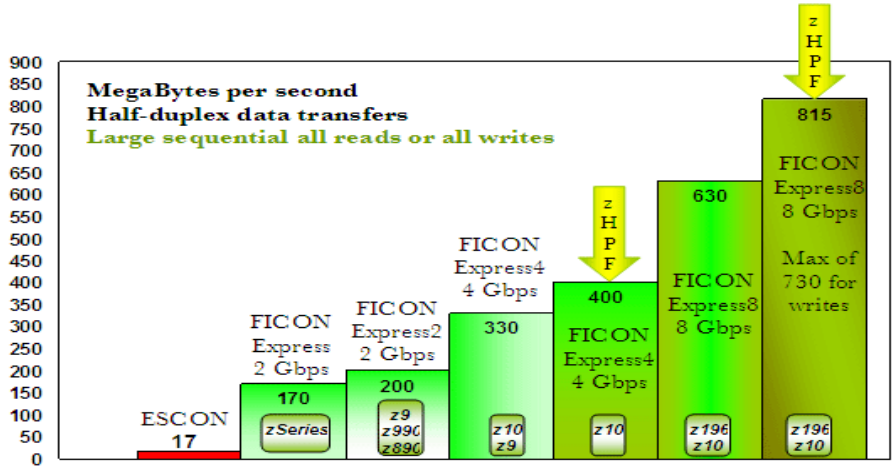


Fig. 6-6. Performance Estimation of FICON. [11]

6.4 HiperSockets

“Mainframe HiperSockets is a technology that provides high speed TCP/IP connectivity within a central processor complex” [12]. “It eliminates the need for any physical cabling or external networking connection between servers running on different LPAR’s “[12].

There are many advantages of HiperSockets performance when compared with Gigabit Ethernet. The throughput increased 1.6 to 2.1 times, comparing a Linux LPAR using HiperSockets to an outboard Linux on Netfinity using OSA-Express GbE [13].

The communication in HiperSockets is generally through the system internal memory of the processor, so servers are connected to form an “internal LAN”. HiperSockets is also called internal QDIO, or IQDIO, since its implementation is based on the OSA-Express Queued Direct I/O (QDIO) protocol [12]. One should remember that z/OS is not the only operating system running on a mainframe host, but also the other operating systems like z/VM and Linux [12].

Let us look at the usage of HiperSockets with an example in **Figure 6-7**.

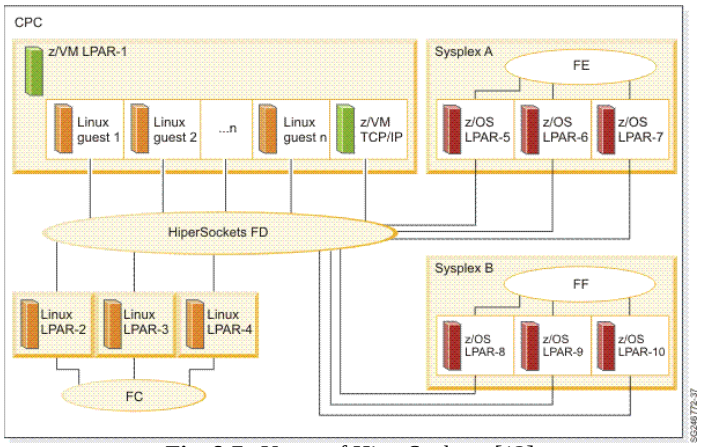


Fig. 6-7. Usage of HiperSockets. [12]

- “HiperSockets with CHPID FC” [12]
- “This particular HiperSockets channel path exclusively serves three Linux systems running in LPAR-2, LPAR-3, and LPAR-4”.[12]
- “HiperSockets with CHPID FD” [12]
- “Connected to this HiperSockets channel path are all servers in the mainframe CPC, which are:
    - The multiple Linux servers running under z/VM in LPAR-1
    - The z/VM TCP/IP stack running in LPAR-1
    - The three Linux servers in LPARs 2 to 4
    - All z/OS servers in sysplex A (LPARs 5 to 7) for non-sysplex traffic”.[12]
- “HiperSockets with CHPID FE” [12]
- “This is the connection used by sysplex A (LPARs 5 to 7) to transport TCP/IP user-data traffic among the three sysplex LPARs”.[12]
- “HiperSockets with CHPID FF”. [12]
- “This is the connection used by sysplex B (LPARs 8 to 10) to transport TCP/IP user-data traffic among the three sysplex LPARs.”[12]

HiperSockets can be customized to accommodate varying traffic sizes. Because HiperSockets does not use an external network, it can free up system and network resources, helping to eliminate attachment costs while improving availability and performance. [14]

Glossary

Book	Ceramic plate with four PUs and two SCs on it
BPA	Bulk Power Assembly – provide power connectivity
BPR	Bulk Power Regulator
CHPID	Channel Path Identifier
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal Oxid Semiconductor; a fabrication technology for modern integrated circuits
CoP	Co-Processor
CP	Central Processor
CPC	Central Processing Complex
DCA	Distributed Converter Assembly
DIMM	Dual Inline Memory Module
eDRAM	Embedded Dynamic Random Access Memory
EMIF	ESCON's Multiple Image Facility
ESCON	Enterprise System Connection
FICON	Fibre Connection
HCA	Host Channel Adapter
HMC	Hardware Management Console
IBF	Internal Battery Feature – to bridge power shortages
ICF	Internal Coupling Facility
IFB	InFiniBand
IFB-MP	InFiniBand – Multiplexer
IFL	Integrated Facility for Linux
Instruction	Callable function of a processor
LPAR	Logical Partitions
MBA	Memory Bus Adapter
MCM	Multi-Chip Module; ceramic plate basically consisting of four PUs and two SC chips
MCU	Memory Control Unit
MDA	Motor Drive Assembly – part of a MRU
MRU	Modular Refrigeration Unit – the z196 air cooling instance
MSA	Motor Scroll Assembly – part of a MRU
OSA	Open Systems Architecture
POR	Power-On-Reset – the "boot" & initialization process of a z196
PU	Processor Unit; the computational core, where operations are executed
QDIO	Queued Direct I/O
RAIM	Redundant Array of Independent Memory
RPQ	Request for Product Qualification
SAN	Storage Area Network
SAP	System Assistance Processor
SC	Storage Control; special full-custom chip mainly used as a shared cache and cache controller between a book's PUs
SE	Support Element – used to manage a mainframe, typically an IBM Thinkpad
SOI	Silicon On Isolator; fabrication technique for better isolation of transistors
WCU	Water cooling Unit – the z196 water cooling instance

zAAP	System z Application Assist Processor
zBX	BladeCenter Extension
zIIP	System z Integrated Information Processor

References

1. IBM Redbooks: *IBM zEnterprise SystemTechnical Guide*, 2010. http://www.redbooks.ibm.com/redbooks/pdfs/sg247833.pdf
2. IBM Redbooks: *IBM zEnterprise System Technical Introduction*, 2010. http://www.redbooks.ibm.com/redbooks/pdfs/sg247832.pdf
3. Dr. Stephan Kammerer: *Mainframe Summit TU Kaiserslautern HW Overview*, IBM Research & Development, 2010.
4. IBM Redbooks: *Introduction to the New Mainframe: z/OS Basics*, 2009. http://www.redbooks.ibm.com/redbooks/pdfs/sg246366.pdf
5. Mike Kahn: *The Beginning of I.T. Civilization – IBMSystem/360Mainframe*, 2004. http://www-07.ibm.com/servers/eserver/includes/download/clipper_mainframe_at40.pdf
6. CISCO: *Mainframe Storage and Networking*, 2008. http://www.iphelp.ru/faq/19/ch01lev1sec5.html
7. IBM: *RS/6000 pSeries, IBM Fibre Channel Planning and Integration: User's Guide and Service Information*, 2002. http://publib.boulder.ibm.com/systems/hardware_docs/pdf/234329.pdf
8. Wikipedia: *ESCON*, 2010. http://en.wikipedia.org/wiki/ESCON
9. IBM Redbooks: *Netfinity to S/390 Connectivity using ESCON*, 1998. http://www.redbooks.ibm.com/redbooks/pdfs/sg245284.pdf
10. IBM Redbooks: *Introduction to IBM S/390 FICON*, 1999. http://www.redbooks.ibm.com/redbooks/pdfs/sg245176.pdf
11. IBM: *IBM System z - I/O Connectivity: FICON/zHPC/CTC*, 2010. http://www-03.ibm.com/systems/z/hardware/connectivity/ficon_performance.html
12. IBM: *IBM Networking on Z/OS*, 2010. http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.znetwork/znetwork_85.htm
13. Jim Goethals, Bob Perrone: *zSeries and z/OS HiperSockets Overview*, 2003. http://www.ibmsystemsmag.com/mainframe/januaryfebruary03/technicalcorner/9920p3.aspx
14. IBM: *IBM Networking Concepts*, 2010. http://www-03.ibm.com/systems/z/hardware/networking/index.html

Redundanz
Seminar: Mainframes Today

Matthias Blume, Julian Hager
deiis@web.de, julianhager@hotmail.com

Database and Information Systems Group
University of Kaiserslautern
Supervisor: Karsten Schmidt

Abstract. Im heutigen, sogenannten Informationszeitalter haben wir es mit einer täglich wachsenden Menge an Informationen zu tun. Sowohl Informationsgröße, als auch die Anzahl der verfügbaren Information wächst sehr schnell. Der Aufwand diese Mengen aktuell zu halten, sie aufzubereiten, zu verarbeiten und in brauchbarer Version bereitzustellen ist vergleichsweise hoch und zeitaufwändig.

Insbesondere wenn man bedenkt, wie schnell sich der Gehalt von Informationen ändern kann und dass der Verarbeitungsgeschwindigkeit als auch dem physikalischen Speicher auf dem Daten abgelegt werden deutliche Grenzen auferlegt sind. Normale Rechensysteme, wie jeder von uns sie daheim stehen hat reichen für die Datenmengen, wie es sie in Firmen oder Instituten zu verarbeiten gilt bei Weitem nicht aus. Einerseits wäre ihre Speicherkapazität zu gering, andererseits wäre die Kommunikation der Komponenten untereinander zu langsam, um die an sie gestellten Kriterien zu erfüllen.

Es bedarf an dieser Stelle spezieller Architekturen, die vor allem auf Performanz und Geschwindigkeit ausgelegt sind. Diese Architekturen beherbergen eine sehr große Anzahl an Komponenten die auf Leistung getrimmt sind und deren Kommunikation aufeinander abgestimmt ist.

Den Datendurchsatz und die Kommunikationsgeschwindigkeit der, im Vergleich zum Prozessor eher langsamen Festplatten lässt sich zum Beispiel durch das Zusammenschalten mehrerer Festplatten gleicher Kapazität zu einem Verbund erhöhen. Mit steigender Komponentenzahl wächst jedoch auch die Anfälligkeit des Systems.

Komponenten können von vorne herein fehlerhaft sein, können ihre Lebensdauer überschreiten oder durch völlig andere Ursachen (wie etwa Erdbeben) ausfallen. Schon als einzelner privater Nutzer ist der Ausfall einer Komponente ein Graus. Alptraumhafte Dimensionen erreicht dieses Beispiel, stellt man sich das minutenlange Stillstehen der Börse vor weil eine

einzelne Festplatte den Dienst verweigert.Diese zu lokalisieren, auszutauschen, das System anschließend wieder zu booten und betriebsfähig zu machen kann zu einer wochenlangen Arbeit ausarten, die unvorstellbare Verluste in der Wirtschaft nach sich ziehen würden.

Um solchen Ernstfällen entgegenzuwirken bedarf es bestimmter Maßnahmen, deren Grundidee es ist, Informationen zu duplizieren und Komponenten mehrfach zu verbauen, sodass keine verheerenden Schäden entstehen und das System seine Arbeit nicht aussetzen muss, weil zusätzlich vorhandene Komponenten für die defekten einspringen. Das Umsetzen dieses Ansatzes bezeichnet man als Redundanz.
(vom lat. redundare – im Überfluss vorhanden sein)

Wir unterscheiden zwei Arten der Redundanz:

1.) die Redundanz der Hardware-Komponenten eines Rechnersystems

2.) die Redundanz der Daten auf einem Rechensystem

Unter Hardware-Redundanz versteht man das zusätzliche Vorhandensein funktional gleicher, oder vergleichbarer Ressourcen (wie etwa Prozessoren oder Festplatten), die im Fall eines Versagens einzelner Komponenten gewährleisten, dass das Gesamtsystem weiterhin in vollem Umfang funktionsfähig bleibt.

Unter Redundanz der Daten versteht man ein mehrfaches Vorhandensein identischer Informationssätze (etwa Anwendungen, Jobs und Information), die auf mehreren Hardwaresystemen (sog. Informationseinheiten) verteilt sind. Eine Informationseinheit gilt als redundant, wenn sie ohne Informationsverlust zu verursachen weggelassen werden kann.

1 Einleitung:

Im Nachfolgenden gehen wir darauf ein, wie Redundanz realisiert wird.

1.1 Definition:

„Der Begriff Redundanz (lat. redundare – im Überfluss vorhanden sein) bezeichnet allgemein in der Technik das zusätzliche Vorhandensein funktional gleicher oder vergleichbarer Ressourcen eines technischen Systems, wenn diese bei einem störungsfreien Betrieb im Normalfall nicht benötigt werden. (...) In aller Regel dienen diese zusätzlichen Ressourcen zur Erhöhung der Ausfall-, Funktions- bzw. Betriebssicherheit.“ [1]

1.2 Warum Redundanz?

Wie bereits erwähnt verlassen sich heutzutage viele große Betriebe auf komplexe IT Systeme um die enorme Menge an Daten zu verwalten, die nicht nur durch die Arbeit im Betrieb selbst, sondern auch durch deren (potenziell viele) Kunden anfallen. Eine Bank, zum Beispiel, muss einerseits intern große Mengen an Daten verwalten, wie etwa für Börsengeschäfte, andererseits entstehen durch deren Bankkunden sehr viele andere Transaktionen. Nichts hiervon darf verloren gehen sonst erleidet die Bank große Verluste, selbst bei nur sehr kurzen Ausfällen. Nicht betätigte Börsentransaktionen bedeuten hohe Geldverluste, unzufriedene Kunden könnten die Bank verlassen.

Damit solche Ausfälle möglichst nie passieren werden die verwendeten Systeme redundant aufgebaut. Das heißt, dass wichtige Teile des Systems mehrfach vorkommen, seien es technische Bauteile wie Festplatten, Kabel, etc. oder Daten, die auf mehreren Festplatten gespiegelt sind. Ursache für solche Ausfälle können unvorhersehbare Ereignisse wie Naturkatastrophen oder Arbeitsunfälle sein. Ein Datacenter, also der Ort an dem alle Daten gespeichert sind (nicht immer im Hauptsitz des Betriebs), könnte ohne spezielle Sicherheitsvorkehrungen z.B. durch einen Stromausfall außer Betrieb gehen. Deshalb gibt es dort Notstromversorgungen, die das Gebäude weiter temporär versorgen.

Bei einer schlimmeren Katastrophe könnte das Gebäude zerstört werden, z.B. durch ein Erdbeben. In einem solchen Fall reichen gebäudeinterne Sicherheitsmechanismen nicht aus. Dieses Problem wird dadurch gelöst, dass es weitere Datacenter an anderen Standorten gibt.

1.3 Begriffe:

Im Folgenden werden einige Begriffe erläutert, die im Zusammenhang mit Redundanz stehen.

1.3.1 High Availability:

High Availability (deu.: Hochverfügbarkeit) beschreibt eine Eigenschaft eines Systems für sehr lange Zeiträume verfügbar zu bleiben. Das beinhaltet auch Robustheit gegenüber Ausfällen einzelner Systemkomponenten, so dass selbst wenn es zu einem solchen Ausfall kommt das System mit hoher Wahrscheinlichkeit (wie z.B. mit der "five-nines" Garantie, also 99.999%) weiterläuft. High Availability kann man garantieren, indem man z.B. alle Single Points of Failure vermeidet. [2]

1.3.2 Single Point of Failure:

Unter einem Single Point of Failure (deu.: einzelne Stelle des Scheiterns) versteht man ein Teil eines Systems dessen Ausfall den Ausfall des gesamten Systems mit sich zieht. So etwas ist in einem System, das Hochverfügbarkeit garantieren soll

unerwünscht. In einem komplexen System müssen solch kritische Komponenten, deren Ausfall zum kompletten Systemausfall führen würden, mittels Analyse des Gesamtsystems identifiziert werden. Ein sehr zuverlässiges System darf sich nicht auf eine einzelne solche Komponente verlassen. [3]

1.4 Umsetzung:

Um den obigen Problemen vorzubeugen werden kritische Systeme redundant aufgebaut. Im folgenden Abschnitt wird einiges zur allgemeinen Umsetzung von Redundanz erläutert.

1.4.1 N+1 Redundanz:

Bei der N+1 Redundanz handelt es sich um ein Konzept, bei dem neben den aktiven Systemkomponenten (N) auch passive Backup-Komponenten (+1) im System integriert sind. Sollte es bei einer der aktiven Komponenten zu einem Defekt kommen, kann diese abgeschaltet und deren Funktion von einer der passiven Komponenten übernommen werden, meist ohne die Performanz des Systems merkbar zu beeinträchtigen. Dieser Vorgang geschieht meist automatisch und wird Failover genannt. [3]

In einer Variation dieses Prinzips sind die zusätzlichen Backup-Komponenten im System aktiv und dienen lediglich dazu das System mit zusätzlichen Komponenten gegen Ausfälle zu schützen. Fällt eine dieser aus, dann reduziert sich auch die Performanz des Systems.

Ein zusätzlicher Vorteil von solch redundanten Systemen ist, dass diese beim laufenden Betrieb gewartet werden können, ihre Verfügbarkeit also nicht dadurch eingeschränkt wird. Selbst bei Hardwareschäden kann das System normal weiterlaufen. Es muss lediglich die defekte Komponente von einem Techniker ausgetauscht werden. Dadurch müssen Betriebe zu keinem Zeitpunkt auf Arbeit, die das System leistet verzichten. [4]

1.4.2 RAID:

Ein Redundant Array of Independant Disks (deu.: Redundante Anordnung unabhängiger Festplatten), früher Redundant Array of Inexpensive Disks (also, kostengünstig anstelle von unabhängig) beschreibt eine Technologie, die eine erhöhte Speicherzuverlässigkeit garantiert.
Ein RAID kann entweder Daten auf mehrere Speichermedien verteilen, oder die Datensätze spiegeln. Es gibt eine ganze Reihe von Möglichkeiten ein RAID-System aufzusetzen (gekennzeichnet durch RAID gefolgt von einer Nummer, also z.B. RAID 0, RAID 1), diese sind Zusammensetzungen dieser Varianten (zB: RAID 01).

1.4.3 Disaster Recovery Plan:

Ein Disaster Recovery Plan (deu.: Notfallwiederherstellungsplan, kurz DRP) beschreibt die Maßnahmen die getroffen wurden, falls ein System zu einem alternativen Standort migriert werden muss. Der DRP befasst sich hauptsächlich mit großen, katastrophalen Ereignissen, die den Zugriff auf ein System im Hauptstandort dauerhaft verhindern. Kann auch Vorkehrungen enthalten, um ein Desaster von vornherein zu verhindern. [2]

2 Redundanz der IBM-Mainframes

Das folgende Kapitel befasst sich mit der Umsetzung der Redundanz in IBM-Mainframes

2.1 Einleitung:

Die Architektur eines Mainframes ähnelt in groben Zügen denen eines gewöhnlichen PCs: Netzteil, Mainboard, Prozessor, Arbeitsspeicher und Festplatte.
Die Quantität- als auch die Qualitätsanforderungen an diese liegen beim Mainframe jedoch in vollkommen anderen Dimensionen. Wie in der Einleitung angesprochen, erfordern gigantische Datenmengen eine wesentlich massivere Rechenleistung und in ihrem kontinuierlichem Erscheinen eine nahezu echtzeitfähige Abarbeitung. Eine der wichtigsten Ansätze ist demzufolge auch der Umstand, dass man in der Lage ist eine Maschine, die solche Daten zu bearbeiten hat kontinuierlich laufen lassen kann und im laufenden Betrieb in der Lage ist, fehlerhafte Komponenten auszutauschen.

Ein Konzept, welches IBM zu diesem Zweck umsetzte, basiert auf einer starken Parallelisierung und Virtualisierung, die eine systemweite Redundanz ermöglicht. Um diese im vollen Umfang zu erläutern müssen wir im folgenden etwas auf die eigentliche Architektur eines Mainframes (z196) eingehen.

2.2 Aufbau eines Mainframes:

Ein Frame besteht aus: [5]

- optional einhängbare Batterien
- Stromversorgungen
- 2 Support-Elementen
- einem bis vier Books
- InfiniBand I/O Konnektoren
- I/O Drawer & I/O Cages

2.2.1 Batterien:

Das optionale Internal Battery Feature (IBF) dient als eine lokale Energiequelle im Fall eines Stromausfalls. Sie sind in der Lage den Mainframe (abhängig von der I/O-Konfiguration) bis zu 13 Minuten mit Energie zu versorgen. Dies ist genug Zeit um Zustände von Prozessen zu sichern und ein fehlerfreies Herunterfahren zu ermöglichen. Die IBF schützen zusätzlich vor Stromspitzen und ihren Folgen.

2.2.2 Stromversorgung:

Für die Energieversorgung des Mainframes sind zwei identische 3-Phasen Power Feeds zuständig. Sollte eines ausfallen, stellt die verbliebene Komponente die Stromversorgung weiterhin sicher.

2.2.3 Support-Elemente:

Um mit dem Mainframe direkt kommunizieren zu können wurden in den Frame zwei Support-Elemente eingefügt, die als direkte Management-Konsole dienen. An ihnen lassen sich umfangreiche Änderungen im System vornehmen. Das zweite Support-Element dient als Ersatzelement, falls das erste aus irgendwelchen Gründen ausfallen sollte. [5]

2.2.4 Books:

Eine entscheidende Rolle spielen die austauschbaren Books, die man sich in etwa als einen einzelnen Rechenblock vorstellen kann, der durch Hinzufügen oder Entfernen das Gesamtsystem leistungsstärker oder -ärmer macht. Ein Frame besteht aus minimal einem, maximal vier Books.

Jedes dieser Bücher verfügt über folgende Komponenten: [6]

- Ein MultiChipModul (MCM)
- Memory DIMMs
- Drei sogenannte Distributed Converter Assemblies (DCAs)
- InfiniBand Host Channel Adapter Fanout Cards mit je 2 Ports.

2.2.4.1 MCM:

Unter dem MCM kann man sich die eigentliche Recheneinheit des Mainframes vorstellen. Es handelt sich dabei um ein Modul, das sechs QuadCore-Mikroprozessor-Chips beherbergt, von denen einer aus Redundanz-Gründen von Werk aus deaktiviert ist und als Ausweichmöglichkeit dient. Sie können jedoch auch bei Bedarf zugeschaltet werden. Je nach Modell variiert die Anzahl der Processing Units (kurz PU) zwischen 20 oder 24 Einheiten. [5]

Fällt eine der PUs aus, werden ihre Aufgaben (CP, IFL, ICF, zAAP, zIIP, oder SAP) dynamisch an eine von zwei Ersatz-PUs abgegeben. Dabei hält sich das System an vorgegebene Regeln zum Ersetzen ausgefallener Komponenten: [7]

1. Tritt der Ausfall auf einem Chip mit vier aktiven PUs auf, nutze die Ersatz-PUs um die ausgefallene PU und die PU die eine Aufgabe mit ihr teilt zu ersetzen.
2. Tritt der Ausfall auf einem Chip mit nur drei aktiven Kernen auf, ersetze die PU, die keine Aufgabe mit einer anderen PU teilt.
3. Sind alle zusätzlichen Kapazitäten belegt, nutze nicht-charakterisierte PUs unter Berücksichtigung der obigen Regeln.

Der Status der Anwendung die auf dem ausgefallenen Kern lief wird dabei migriert und auf dem neu zugewiesenen Kern fehlerfrei fortgeführt ohne dass das Betriebssystem intervenieren muss. Ein weiteres nennenswertes Feature der PUs ist die Mirror-Funktion, die eine Großzahl der Instruktionen zweimal ausführt um deren Integrität und Richtigkeit zu prüfen.

Sind keine Ausweichkerne mehr vorhanden, oder wird die Arbeitsweise des Systems aufgrund eines Capacity on Demand verändert, meldet die Remote Support Facility IBM die Notwendigkeit des Ersetzens einer MCM. Diese Möglichkeiten reduzieren die Wahrscheinlichkeit eines Systemstillstands und erlauben präventive Maßnahmen.

2.2.4.2. Memory DIMM:

Der physikalische Speicher eines Books ist wie folgt aufgebaut:
Jedes Book verfügt über 10 bis 30 DIMMs, die entweder vollständig oder teilweise belegt sind. Das absolute Speicherminimum für ein Book beträgt 40 GB RAM, von denen 16 bereits für die Hardware-System-Area reserviert sind.
Ähnlich wie bei der MCM verfügt jedes Book über zusätzlichen, deaktivierten Speicher, der auf Anfrage dynamisch zugeschaltet werden kann.

Ein Großteil des verbauten Speichers (20%) wird verwendet um ein neues Prinzip namens RAIM (Redundant Array of Independent Memory) zu realisieren. RAIM erlaubt, seinem Namensgeber, dem RAID nicht unähnlich, höhere Datentransferraten und, im Fall des Defekts einzelner RAM-Bausteine, eine höhere Datenverfügbarkeit. RAIM ist in der Lage Defekte im DRAM, Socket, Memory-Channel oder DIMM zu erkennen und während der Laufzeit zu korrigieren. [6]
Die Realisierung des RAIMs beziehungsweise das Management der DIMM-Blöcke erfolgt über die sogenannten Memory-Control-Units (MCU).

Die DIMMs sind über drei Memory-Control-Units mit dem MCM verbunden. Jede dieser MCUs nutzt fünf Kanäle. Den vier DIMM-Blöcken ist über den fünften Kanal der MCU ein zusätzlicher DIMM-Block zugeschaltet, dessen Zweck es ist, fehlerhafte Speicherkomponenten zu lokalisieren, sie dynamisch zu ersetzen oder eventuell fehlerhafte Ergebnisse zu korrigieren. Diese Parität führt zu einer vollständig fehlertoleranten N+1 Redundanz. [5]

2.2.4.3. Distributed Converter Assemblies:

Drei sogenannte Distributed Converter Assemblies (DCAs) versorgen die Books mit ausreichend Energie. Der Verlust eines DCAs fällt aufgrund der zusätzlich vorhandenen Einheit nicht weiter ins Gewicht. In diesem Zusammenhang lohnt es sich auch die Voltage Transformation Modules zu nennen.
Die n+2 Redundanz der VTMs stellt sicher, dass durch zu geringe Spannung einige Prozessoren ausfallen und auf den verbleibenden Prozessoren eine vermehrte Prozessorlast auftritt. [5]

2.2.4.4. InfiniBand Host Channel Adapter Fanout Cards:

Für eine schnelle Verbindung mit den I/O-Drawers oder den Coupling Facilities sorgen die Host Channel Adapter Fanout Cards (HCA), von denen jedes Buch bis zu acht Stück besitzt.
Jede dieser Fanout Cards verfügt über zwei Ports, die bidirektionalen Verkehr erlauben. Somit werden bis zu 16 Verbindungen pro Book ermöglicht.
Redundanz erreicht man hier über die Kopplung der Infiniband I/O an Infiniband Multiplexer (IFB-MP).

Diese ordnen den Fanout-Karten Domänen zu und erlauben im Fall des Versagens einer Fanout-Karte eine Weiterleitung des Datenstroms an die restlichen Domänen. Zum besseren Verständnis könnte man sich an dieser Stelle eine Umleitung von einer Autobahn auf eine Ausweichstrecke vorstellen. Diese Ausweichstrecke kann ein anderes Book sein, kann aber auch eine andere Maschine sein, die Teil eines Parallel-Sysplex ist.
Fällt eine der IFB-MP-Karten aus, sind alle I/O-Karten in der betroffenen Domäne unbrauchbar, bis die nötigen Reparaturen abgeschlossen wurden. [5]

Dieses Konzept ermöglicht es im Übrigen auch, Books im laufenden Betrieb hinzuzufügen oder zu entfernen. (Enhanced Book Availability) Die Gesamtlast des Systems verändert sich hierbei nicht, sofern die übrigen Books über genug freie Ressourcen verfügen um das fehlende Book kurzzeitig zu ersetzen.
Prozesse, Zustände und Aufgaben werden über die verbliebenen Books verteilt und dort fortgeführt.

2.2.5 Infiniband I/O-Konnektoren

Die Infiniband I/O-Konnektoren dienen dem internen Datentransfer zwischen den Books und den I/O-Cages/Drawern. Ihre Funktionsweise ähnelt denen der Fanout-Karten der Books.

2.2.6 I/O Drawer & I/O Cages:

I/O-Feature-Cards bieten mehr als genug Ports um den Mainframe an externe Medien, Netzwerke oder andere Mainframes zu verbinden. I/O-Karten werden entsprechend den Konfigurationsregeln des Mainframes in die entsprechenden I/O-Cages/Drawer gesteckt.Für Jede Art von Kanal oder Link-Typ existieren I/O-Karten-Typen. [6]

I/O-Kanäle sind Teil des Kanal-Subsystems (CSS).
Sie stellen Datenverbindungen zwischen Servern, externen Kontrolleinheiten (CUs) und Geräten oder Netzwerken her. Kommunikation erfolgt über den InterSystem-Kanal (ISC4), den integrierten Clusterbus (ICB4) für Coupling via Infiniband oder Kanal-zu-Kanalverbindungen. (CTC). Auch hier gilt, dass etwaig fehlerhafte Komponenten durch andere ersetzt werden um somit eine relativ konstante Übertragungsrate zu erreichen. [5, 7]

2.3 Capacity on Demand:

Wie zuvor erwähnt sind in Books mehr Komponenten verbaut als aktiviert.
Neben den oben genannten Redundanz-Anforderungen kommt eine weitere Idee namens Capacity on Demand zum Tragen.

CoD ermöglicht temporäres Freischalten ungenutzter Kapazitäten sofern die Notwendigkeit dafür entstehen sollte. Den Schlüssel zur Aktivierung zusätzlicher Ressourcen stellt der PR/SM-Manager dar. Es handelt sich hierbei um einen Prozess der alle Komponenten als ein einziges SMP-System verwaltet und genau weiß, welche Ressourcen wofür verwendet werden.
Er hat zudem einen Überblick über die Gesamtheit der installierten Hardware; genutzt oder ungenutzt. [5]

Via eines speziellen Codes, den man bei IBM erwerben kann, werden zusätzliche Komponenten aktiviert. Der PR/SM-Manager skaliert das System entsprechend der neu vorhandenen Ressourcen und weist dazugewonnen Speicher entsprechend neu zu. Dementsprechend unterstützt der PR/SM-Manager unter anderem die zuvor angesprochene Enhanced Book Aviability. [8]

Es existieren drei Arten von CoD, die für verschiedene Zwecke gedacht sind:

- Capacity Backup: Aktivieren zusätzlicher Ressourcen für kritische Backup-Prozesse
- Capacity for planned Events: Aktivieren zusätzlicher Ressourcen für einen begrenzten Zeitraum
- On/Off CoD: Dauerhafte Nutzung zusätzlicher Ressourcen. Zusätzliche Ressourcen können auf Wunsch wieder deaktiviert werden, müssen aber nicht.

3 Verteilte Strukturen:

Um Sicherheit in einem System zu schaffen kann es aufgeteilt werden. Im folgenden Abschnitt wird einiges zur Umsetzung erläutert.

4.1.Parallel Sysplex:

Unter einem Parallel Sysplex (Systems Complex) versteht man mehrere IBM Mainframes, die zu einem großen Cluster zusammengefasst sind und somit ein großes System bilden. Das ganze System wird dabei von nur einem einzigen z/OS gesteuert. In einem Sysplex können bis zu 32 Systeme miteinander vernetzt werden. Im Zusammenhang mit Redundanz wird der Parallel Sysplex zur verteilten Datensicherung und Disaster Recovery verwendet. Hierbei ist die Datensynchronisation von großer Bedeutung.

In einem Sysplex enthalten sind folgende Komponenten:

- Ein Sysplex Timer der die Uhren aller zugehöriger Systeme synchronisiert
- Global Resource Serialization (GRS), welches mehreren System den parallelen Zugriff auf dieselben Ressourcen zur selben Zeit, gegebenenfalls serialisiert, um exklusiven Zugriff zu versichern.
- Cross System Coupling Facility (XCF), welches Kommunikation über peer to peer erlaubt
- Couple Data Sets (CDS)
- Wichtige Komponenten eines Parallel Sysplex:
- Coupling Facility (CF oder ICF) hardware, die
- Sysplex Timer oder Server Time Protocol zur Synchronisation der Uhren
- Schnelle, hochwertige und redundante Verbindung
- Software (Betriebssysteme und oft auch Middleware wie etwa DB2)

Eine Coupling Facility kann einerseits ein spezialisiertes externes System sein (z.B. ein minimal ausgestatteter Mainframe, der speziell nur mit Coupling Facility Prozessoren konfiguriert ist) oder integrierte Prozessoren auf den Mainframes , die als Internal Coupling Facilities (ICFs) konfiguriert sind. Ein Parallel Sysplex hat mindestens zwei CFs, bzw. ICFs, der Redundanz wegen.

4.2. Geographically Dispersed Parallel Sysplex

Geographically Dispersed Parallel Sysplex (GDPS) ist eine Erweiterung des Parallel Sysplex. Hierbei sind die Mainframes räumlich getrennt, sie können sich z.B. in verschiedenen Städten befinden. Es gibt für GDPS sowohl eine Single-Site als auch eine Multiple-Site Konfiguration

GDPS/HyperSwap-Manager: Eine synchrone peer to peer Technologie (PPRC), die innerhalb der Datacenter verwendet wird. Die Daten auf dem primären Speicher werden auf den sekundären Speicher kopiert. Im Fall eines Fehlers auf der primären Speichereinheit schaltet das System automatisch auf die sekundäre um, meist ohne den laufenden Betrieb des Systems zu stören.

GDPS/PPRC: Ist eine synchrone Datenspiegelungstechnologie (PPRC), welche auf Mainframes die bis zu 200 km voneinander entfernt sind, verwendet werden kann. Sollte ein System oder eine Speichereinheit ausfallen setzt die Wiederherstellung automatisch ein.

GDPS/XRC: Eine asynchrone Extended Remote Copy (XRC) Technologie, bei der die Distanz nicht eingeschränkt ist. Beim Kopieren zwischen zwei Standorten entsteht nur eine kleine Verzögerung so dass bei einem Ausfall Daten von nur ein paar Sekunden verloren werden. Sollte es zu so einem Ausfall kommen, so muss der Benutzer ein Wiederherstellungsverfahren einleiten. Hierbei werden automatisch die Daten vom sekundären Standort wiederhergestellt und das System umkonfiguriert.

GDPS/GM: Eine asynchrone IBM Global-Mirror-Technologie ohne Einschränkung auf Distanz. Dieses System dient zur Wiederherstellung nach einem totalen Systemausfall an einem Standort.

GDPS/MGM&GDPS/MzGM: Diese Konfiguration ist für Systeme gedacht, die auf mehr als zwei Standorten verteilt ist.

5. Quellenangaben:

[1] http://de.wikipedia.org/wiki/Redundanz_%28Technik%29 (15.11.2010)
[2] Redbook: Useful Disaster Recovery Definitions and Concepts
[3] http://en.wikipedia.org/wiki/N%2B1_redundancy (17.11.2010)
[4] Redbook: The IBM Mainframe Today: System z Strengths and Values
[5] Redbook: IBM System z10 Business Class Technical Overview
[6] Redbook: IBM zEnterprise System Technical Introduction
[7] Redbook: IBM System z10 Enterprise Class Technical Introduction
[8] IBM System z10 Capacity on Demand
[9] IBM zEnterprise System Technical Guide
[10] http://www.sdisw.com/vm/dasd_capacity.html (30.11.2010)
[11] http://www.redbooks.ibm.com/abstracts/tips0054.html

Virtualisierungstechnologien

Seminar: Mainframes Today

Thomas Psota, Christopher Schank

Database and Information Systems Group
University of Kaiserslautern

1 Virtualisierung

In der Informatik ist der Begriff der **Virtualisierung** schwer zu definieren. Eine mögliche Definition wäre: Virtualisierung bezeichnet Methoden, die es erlauben, Ressourcen eines Computers (insbesondere im Server-Bereich) zusammenzufassen oder aufzuteilen. In dieser Arbeit beschäftigen wir uns mit der sogenannten Hardware-Virtualisierung. Hardware-Virtualisierung versteckt die physische Hardware vor dem Benutzer und stellt stattdessen eine abstrakte Betriebsplattform zur Verfügung. Dadurch können zum Beispiel mehr virtuelle Ressourcen erzeugt werden, als tatsächlich physisch vorhanden sind, um die Auslastung der reellen Hardware zu erhöhen. Bei der Hardware-Virtualisierung wird ein komplettes System so emuliert, dass Gast-Betriebssysteme ohne Veränderungen lauffähig sind. Im Idealfall ist sich das Gastsystem nicht bewusst, dass es auf virtueller Hardware läuft. Ein Gastsystem soll keinen Zugriff auf die Ressourcen eines anderen Gastes haben, d.h. es soll komplett von der eigentlichen Hardware isoliert werden. Es existieren viele Gründe für Virtualisierung, zum Beispiel können mit Hilfe von Virtualisierung Anwendungen benutzt werden, die vom eigentlichen Betriebssystem nicht unterstützt werden, oder neue Betriebssysteme und Software können evaluiert und getestet werden, ohne das Produktivsystem zu gefährden. Ein weiterer Grund für Virtualisierung ist die sogenannte Serverkonsolidierung, d.h. dass mehrere dedizierte Server durch einen einzelnen physikalischen Server ersetzt werden, während sich aus logischer Sicht für die Außenwelt nichts ändert. Dadurch können Kosten gespart werden, da weniger Server gekühlt und mit Strom versorgt werden müssen und es wird eine höhere Auslastung erreicht. [1 , 2]

Die Software, die diese Virtualisierung kontrolliert, ist der sogenannte **Hypervisor** oder auch **Virtual Machine Monitor**. Der Hypervisor stellt den Gast-Betriebssystemen eine virtuelle Betriebsplattform zur Verfügung. Robert P. Goldberg [3] teilt Hypervisors in zwei Klassen ein:

Typ-1: Der Hypervisor läuft direkt auf der Hardware und überwacht die Gastsysteme, d.h. Gastsysteme laufen zwei Ebenen über der Hardware. Beispiele sind *IBMs PR/SM, XEN* und *VMWare*.

Typ-2: Der Hypervisor läuft seinerseits innerhalb eines anderen Betriebsystems, d.h. die Gastsysteme laufen auf der dritten Ebene über der Hardware. Beispiele für einen Typ-2 Hypervisor sind *Microsofts Virtual Server* und *Virtual PC*.

Der erste Hypervisor, der eine sogenannte *Volle Virtualisierung* bot, war der *CP-40*, ein Forschungsprojekt von IBM aus dem Jahr 1967. *Volle Virtualisierung* bedeutet, dass im Gegensatz zur Paravirtualisierung das Gastbetriebssystem unverändert benutzt werden kann, d.h. der Hypervisor simuliert einen kompletten Satz Hardware. Mit *CP-40* war es also erstmals möglich, mehrere unabhängige Betriebssysteme gleichzeitig zu verwenden. Mit dem *IBM System/360-67* kam 1966 das erste Rechnersystem auf dem Markt, welches diese Virtualisierungstechnik enthielt. Heute unterstützen alle IBM Mainframes Virtualisierung, wobei anzumerken ist, dass die *zSerie* von IBM immer noch rückwärtskompatibel mit dem *IBM System/360* ist. [4]

Die abstrakte Betriebsplattform, die der Hypervisor zur Verfügung stellt, wird bei IBM **LPAR** (**Logical Partition**) genannt und Beispielweise bei *XEN* **Domain**. Ein anderer weit verbreiteter Begriff ist **VM** oder **Virtual Maschine**. Mithilfe des Hypervisors können sich mehrere LPARs die vorhandenen physischen Ressourcen teilen, wie zum Beispiel Speicher, Festplatten, CPU und Netzwerk. Dabei haben die einzelnen LPARs aber keinen Zugriff auf die Ressourcen anderer LPARs, d.h. jede LPAR hat ihren eigenen Speicherbereich und kann nicht auf den Speicherbereich einer anderen LPAR zugreifen (außer über den Umweg über einen Prozess auf der anderen LPAR). Diese Isolation der einzelnen Partitionen ist sehr wichtig, damit zum Beispiel das Produktionssystem nicht durch ein Testsystem gefährdet werden kann. Innerhalb einer LPAR kann ein beliebiges, kompatibles Betriebssystem laufen oder auch ein weiterer Hypervisor (zum Beispiel IBMs z/VM). [5]

Lange Zeit war man der Meinung, dass die *x86* Architektur nicht virtualisiert werden könnte. Der Grund dafür war das Privilegiensystem von *x86*, welches vier unterschiedliche Privilegienebenen zur Verfügung stellt, die sogenannten *Ring 0* bis *Ring 3*. Während normale Anwendungen im *Ring 3* laufen, muss ein Betriebssystem Zugriff auf den *Ring 0* haben. Um *x86* also zu Virtualisieren, muss man den Hypervisor unter das Betriebssystem platzieren und darf diesem keinen Zugriff auf den *Ring 0* gewähren. Das Problem dabei ist, dass einige *Ring 0* Instruktionen nicht virtualisiert werden können, da sie eine andere Semantik besitzen, wenn sie nicht im *Ring 0* ausgeführt werden. Diese Instruktionen müssen also abgefangen und übersetzt werden. Dieses Problem wurde 1998 von der Firma VMware gelöst mithilfe der sogenannten *Binary Translation*-Technik. Diese Technik wandelt die sensitiven *Ring 0* Instruktionen in eine Reihe anderer Instruktionen um, die den gewünschten Effekt auf der virtuellen Hardware haben. [6]

Seit 2006 sind von Intel und AMD Prozessoren erhältlich, die **Hardware As-**

sisted Virtualization unterstützen, d.h. dass Virtualisierung direkt von der Hardware unterstützt wird. Bei AMD heißt diese Technologie **AMD-V** und bei Intel **VT-x**. Diese Technologien erlauben es dem Hypervisor unter dem *Ring 0* zu laufen, während die privilegierten *Ring 0* Aufrufe automatisch abgefangen und an den Hypervisor weitergeleitet werden. Dadurch sind Techniken wie *binary translation* überflüssig.

2 XEN

XEN ist ein Typ-1 Hypervisor für *IA-32*, *x86-64*, *Itanium* und *ARM*-Architekturen, der von der University of Cambridge entwickelt wurde. Projektleiter war Ian Pratt, der später die Firma XenSource Inc. gegründet hat, die jetzt das Open Source Projekt XEN unterstützt und die Enterprise-Editionen von XEN vertreibt. Der erste öffentliche Release von XEN war am 9. September 2003. *XenServer* ist als freie Open Source Software erhältlich.

XEN unterstützte zuerst nur die sogenannte *Paravirtualisierung*, d.h. das Gastbetriebssystem musste angepasst werden, um in der virtuellen Umgebung lauffähig zu sein. Durch die Mithilfe von Intel und AMD unterstützt XEN ab der Version 3.0 auch den Betrieb unmodifizierter Gast-Betriebssysteme. Dadurch können auch proprietäre Betriebssysteme wie *Microsofts Windows* unter XEN laufen.

XEN unterstützt das Migrieren von virtuellen Maschinen von einer reellen Hardware zur Anderen ohne Unterbrechung der Verfügbarkeit übers Netzwerk. Um das zu erreichen, kopiert XEN stückweise den Hauptspeicher zur Zielplattform, ohne die Ausführung des virtuellen Systems zu unterbrechen. Nur zur finalen Synchronisierung wird die Ausführung für 60-300 ms unterbrochen. Durch diese Technik kann neue Hardware aufgebaut werden, ohne eine Unterbrechung im Betrieb zu erzeugen. [7]

Bei XEN werden die virtuellen Maschinen als *Domänen* bezeichnet. Das erste Gast-Betriebssystem wird als *Domain 0* bezeichnet und wird automatisch beim Systemstart erzeugt. Die *Domain 0* hat spezielle Verwaltungsprivilegien, um die physische Hardware zu erreichen und mit den anderen Domänen zu interagieren. Administratoren können sich auf der *Domain 0* anmelden und alle anderen *Domänen* verwalten. Alle anderen Gastsysteme werden als *Domain U* Gäste bezeichnet oder als unpriviligierte Gäste. *Domain U PV Gast* bezeichnet hierbei die Gäste, die mit Hilfe von Paravirtualisierung laufen, typischerweise modifizierte Linux, Solaris, FreeBSD oder andere UNIX Betriebssysteme. Voll Virtualisierte virtuelle Maschinen werden als *Domain U HVM Gast* bezeichnet und benutzen unmodifizierte Betriebssysteme, wie zum Beispiel Windows. *Domain U PV Gäste* sind sich bewusst, dass sie keinen direkten Hardwarezugriff haben und dass noch andere virtuelle Maschinen auf dem System laufen, während *Domain U HVM Gäste* nicht wissen, dass sie sich Ressourcen mit anderen virtuellen Maschinen teilen. [8]

3 VMware

VMware wurde 1998 von Diane Greene, Mendel Rosenblum, Scott Devine, Edward Wang und Edouard Bugnion gegründet. Im Mai 1999 wurde ihr erstes Produkt, *VMware Workstation*, veröffentlicht. 2001 versucht sich VMware mit *VMware GSX Server* und *VMware ESX Server* erstmals im Serverbereich. 2003 veröffentlicht VMware *VMware Virtual Center* und *VMotion*. Ab 2004 unterstützen die VMware-Produkte die *64bit-Architektur*.

Die VMware-Produkte stellen dem Gastsystem eine virtuelle Hardware zur Verfügung, emulieren dabei aber nicht jede Prozessorinstruktion. VMware emuliert keine Hardware, die nicht physikalisch vorhanden ist, d.h. das Gastsystem muss auf der gleichen Prozessorarchitektur arbeiten können wie das Hostsystem. Die VMWare-Produkte versuchen so weit wie möglich, Code direkt auf der CPU auszuführen und falls dies nicht möglich ist, versucht die Software den Code dynamisch umzuschreiben (von VMware wird dieser Prozess als "binary translation" bezeichnet). Auf diese Art arbeiten die VMware-Produkte schneller als normale Emulatoren und trotzdem können ummodifizierte Betriebssysteme für die Gäste verwendet werden. [9]

VMware Workstation ist ein Typ-2 Hypervisor und erlaubt dem Benutzer, mehrere virtuelle Systeme auf einem physischen System auszuführen. Dabei ist *VMware Workstation* auf x86- oder x86-64 kompatible Gastbetriebssysteme beschränkt. [10]

VMware ESX ist ein Typ-1 Hypervisor und ist für den Geschäftsgebrauch gedacht. Dadurch, dass *VMware ESX* direkt auf der Hardware läuft, ist die Software wesentlich schneller als zum Beispiel *VMware Workstation*. *VMware ESX* enthält unter anderem noch *VMotion* und Storage VMotion, welches ein Migrieren einer VM im laufenden Betrieb erlaubt. *VMotion* migriert virtuelle Maschinen in drei Etappen: Zuerst wird überprüft, ob die virtuelle Maschine sich in einem stabilen Zustand befindet. In einer zweiten Etappe werden alle Statusinformationen (Hauptspeicher, Register und Netzwerkverbindungen) auf das Zielsystem kopiert. Danach nimmt die virtuelle Maschine ihre Arbeit wieder auf. Anzumerken ist hierbei, dass sich die virtuelle Maschine während des Transfers in einem Ruhezustand befindet (also nicht Verfügbar ist) und dass bei diesem Vorgang nicht die virtuellen Festplatten kopiert werden, der Speicher befindet sich also immer noch auf dem alten System. Mit *Storage VMotion* können auch die virtuellen Festplatten migriert werden. *VMware ESX* beinhaltet auch die sogenannte *Service Console*, ein kleines Betriebssystem zur Verwaltung der virtuellen Maschinen. [11 , 12]

4 Microsoft Hyper-V

Microsoft Hyper-V ist ein von Microsoft entwickeltes Virtualisierungssystem für kommerzielle x86/64 Prozessoren und der Nachfolger zu Microsofts *Virtual Server*.

Hyper-V ist Hypervisor basiert und unterstützt isolierte Partitionen auf denen Betriebssysteme installiert werden können. Der Hypervisor muss mindestens eine *Parent Partition* besitzen auf der ein Windows Server 2008 ausgeführt wird. Dort befindet sich auch der *Virtualisierungs-Stack*, welcher direkten Zugriff auf die Hardware besitzt. Auf der Parent Partition lassen sich Child Partitionen anlegen, welche die Gastbetriebssysteme ausführen.

Die virtualisierten Partitionen haben keinen direkten Zugriff auf den physischen Prozessor. Reelle Interrupts werden ebenfalls nicht abgehandelt. Stattdessen besitzen sie eine virtuelle Sicht des Prozessors und laufen in einer *Guest Virtual Address*, welche nicht notwendigerweise dem gesamten *Virtuellen Address Bereich* des Systems entsprechen muss. Der Hypervisor kann festlegen, welche und wie viele Prozessoren einer *Child Paritition* zugewiesen werden. Die Prozessor Interrupts werden vom Hypervisor abgehandelt und dann an die zuständige Partition weitergeleitet. Die Übersetzung von Gast Virtual-Address Bereichen, lässt sich durch die von den Prozessoren zur Verfügung gestellten Methoden zur Address-Übersetzung beschleunigen.

Child Partitionen haben ebenso keinen direkten Zugriff auf Hardware Ressourcen, stattdessen wird ihnen ihre Hardware virtualisiert. Jede Anfrage von diesen virtuellen Ressourcen wird durch den *VMBus* an die Parent Partition weitergeleitet, welche die Anfragen verwaltet. Der VMBus ermöglicht auch Kommunikation zwischen den einzelnen Child Partitionen.

Die von Microsoft entwickelte Virtualisierungstechnologie *Enlightened I/O* ermöglicht die Hardware-nahe Implementierung von High Level Kommunikationsprotokollen für Netzwerk, Datenspeicherung, Grafiksysteme, etc. auf dem VMBus, damit jegliche Virtuelle Hardware Ebene umgangen werden kann. Das macht die Kommunikation zwar effizienter, verlangt aber, das Enlightened I/O vom Gastbetriebssystem unterstützt wird. Bis jetzt bieten nur *Windows Server 2008*, *Windows Vista*, *Red Hat Enterprise Linux* und *SUSE Linux* Unterstützung für Enlightened I/O.

Unterstützte Gastbetriebssysteme auf Hyper-V sind:

- Windows Server 2000 / 2003 / 2008
- Windows XP / Vista / 7
- Red Hat Enterprise Linux / SUSE Linux
- Theoretische Unterstützung für Linux Kernel 2.6.32 Systeme

5 System z PR/SM

System z Processor Resource/System Manager (PR/SM) ist ein Typ 1 Hypervisor der in jedem *IBM System z Modell* integriert ist. Seine Aufgabe ist es physische Resourcen in virtuelle Resourcen zu transformieren, damit möglichst viele logische Partitionen die selben physischen Resourcen mitbenutzen können.

PR/SM teilt dabei die physischen Systemresourcen (dediziert oder geteilt) in isolierte logische Partitionen. Jede logische Partition operiert als ein eigenes System auf dem sein eigenes Betriebssystem läuft. Auf den neuesten System z Modellen sind bis zu 60 logische Partitionen anlegbar. Die darauf möglichen ausführbaren Gastsysteme sind: *z/VM*, *z/OS*, *Linux*, *Transaction Processing Facility* und *z/VSE*.

Die einzelnen logischen Partitionen kriegen von PR/SM Prozessoren und I/O Ressourcen zugeteilt. Diese können sowohl dediziert einer logischen Partition zugewiesen werden, als auch mit einem *shared*-Zugriff von mehreren Partitionen geteilt werden. Der Speicher wird von PR/SM nur dediziert an logische Partitionen verteilt, kann aber bei Bedarf dynamisch angepasst werden. Operatoren können während der Laufzeit die Benutzung von Prozessoren, welche von mehreren aktiven Logischen Partitionen zur gleichen Zeit geteilt werden, limitieren und modifizieren. Ein System mit nur einem Prozessor, kann ebenfalls mehrere Logische Partitionen erstellen, da PR/SM einen internen Rechenzeitverteiler besitzt, der eine Portion der Prozessorzeit, an jede Logische Partition verteilen kann, ähnlich der Vorgehensweise eines Betriebssystems, welches eine Portion seiner Prozessorzeit an verschiedene Prozesse, Threads oder Aufgaben alloziiert. Dedizierte Resourcen gehören der zugewiesenen logischen Partition wirklich, während logische Partitionen mit geteilten Resourcen den Anschein erhalten, als würden ihnen diese Resourcen alleine gehören, in Wirklichkeit werden diese aber mit anderen Partitionen geteilt. Vor ihrer Aktivierung, wird jeder logischen Partition ihr *Central Storage* und ihr *Expanded Storage* zugeteilt. Diese entspricht dem Festplattenspeicher des System z und kann nicht von mehreren Logischen Partitionen gleichzeitig geteilt werden.

Das Sharen von Prozessoren und I/O Resourcen auf Hardware-Ebene wird durch *Multiple-Image Faciliy* (MIF) und *Multiple-Channel Subsystem* (MCSS) ermöglicht. Diese Technologien sorgen dafür, dass die I/O Kanal Pfade und Subsets von I/O Geräten, die mit diesen Kanälen verbunden sind, von verschiedenen logischen Partitionen geteilt werden können.

PR/SM unterstützt Kryptopraphie auf Hardware-Ebene durch *Crypto Express2* bzw. *Crypto Express3* so wie *Central Processor Crypto Assist Functions* (CPACF). [15 , 16]

6 z/VM

Typischerweise existieren 3 Möglichkeiten einen IBM Mainframe zu betreiben.

+ Der Basisbetrieb der das komplette physische System des Mainframes als ein System verwendet. (Auf neueren Mainframes wird dieser Modus nicht mehr verwendet.)
+ Der LPAR Modus, in dem der Mainframe in mehrere logische Partitionen aufgeteilt wird, auf denen einzelne Betriebssysteme installiert werden können.
+ Der z/VM Modus, der es ermöglicht auf einer LPAR, mehrere Gastbetriebssysteme laufen zu lassen. Auf diesen Modus soll hier näher eingegangen werden.

z/VM ist ein Betriebssystem für die *IBM System z* Plattform, welches eine sehr flexible Test - und Produktionsumgebung bereitstellt. Die z/VM Implementierung der IBM Virtualisierungstechnologie ermöglicht es voll-funktionsfähige Betriebssysteme wie Linux, z/OS und andere als "Gäste" auszuführen. z/VM stellt hierzu jedem Benutzer eine individuelle Arbeitsumgebung, genannt *Virtual Machine*, zur Verfügung. Die Virtuelle Maschine simuliert die Existenz eines dedizierten realen Systems, inklusive Prozessorfunktionen, Speicher, Netzwerken und I/O Resourcen. Es ist also möglich, mehrere verschiedene Linux bzw. z/OS Systeme auf einer einzigen z/VM laufen zu lassen, also auf einem einzigen physischem System.

Eine virtuelle Maschine benutzt zwar reale Hardware Ressourcen, verwendet aber eine virtuelle Addressierung, sie kennt also nur "virtuelle Hardware" welche auch nicht notwendigerweise in der realen Welt existieren muss.

Festplatten in Mainframes werden üblicherweise als *Direct Access Storage Devices* (DASD) bezeichnet. Der Zugriff auf die DASD wird dabei vom *z/VM Control Program* geregelt.

Gastbetriebssysteme haben sowohl Netzwerkzugriff durch echte Netzwerkressourcen des Systems, als auch ein internes virtuelles Netzwerk mit anderen Gastbetriebssystemen. Dieses virtuelle Netzwerk ist dabei besonders schnell, da es sich auf Hauptspeicherebene befindet.

Die Verwendungszwecke solcher VMs können sehr vielseitig sein, u.a. Testen von neuen Programmen oder Betriebssystemen, das Erstellen von unabhängigen Trainingsumgebungen, sowie die Möglichkeit des gleichzeitigen Wartens von Betriebssystemen ohne dabei Produktionsabläufe unterbrechen zu müssen.

Die Sicherheit von z/VM wird gewährleistet durch externe Sicherheitsmanager sowie verschiedene Privilegienklassen, die den Benutzern vom Operator zugewiesen werden können.

Mögliche Gastbetriebssysteme für z/VM sind *Linux for zSeries*, *z/OS* und *z/VM* selbst. Linux profitiert hierbei von speziell darauf ausgelegten Optimierungen der z/VM.

6.1 Stärken von z/VM:

Integrität und Sicherheit:

+ z/VM unterstützt die Verwendung der Kryptographischen Einrichtungen von unterstützten IBM Servern.
+ IBM garantiert, dass jegliche aufgedeckte Verstösse gegen die Integrität durch unauthorisierte Anwendungen korrigiert werden.
+ Integrierte Zugriffskontrolle und Authentifizierungsdienste können durch einen *Externen Sicherheits Manager* (ESM), so wie beispielsweise *RACF*, erweitert werden.

6.2 Verfügbarkeit und Zuverlässigkeit:

+ Anwendungs Wiederherstellung

z/VM stellt Dienste zur Verfügung, welche unvollständige Interaktionen mit Ressourcen Managern wiederherstellen.

+ Automatisierbare Operationen

System Management-Abläufe können komplett automatisiert werden und sind dabei frei-konfigurierbar.

+ Transparente Synchronisierung von duplizierten Daten

Das Synchronisieren von primären und Backup-Daten ist komplett transparent einsichtbar. Im Falle eines Fehlers kann automatisch auf die Backup Kopie der Daten gewechselt werden.

+ Dynamische Rekonfiguration

Viele Aspekte von z/VM lassen sich dynamisch während der Laufzeit neu Konfigurieren, es entsteht dadurch kein Bedarf eines aufwändigen Reboots des gesamten Systems.

+ Konnektivität

z/VM Systeme können miteinander verbunden werden, für eine verbesserte Server Verfügbarkeit.
[16 und 17]

7 Quellenangabe

[1] http://de.wikipedia.org/wiki/Virtualisierung_(Informatik)
[2] http://en.wikipedia.org/wiki/Platform_virtualization
[3] Architectural Principles for Virtual Computer Systems, February 1973, Goldberg, Robert P.
[4] http://en.wikipedia.org/wiki/Hypervisor
[5] http://en.wikipedia.org/wiki/LPAR
[6] Understanding Full Virtualization, Paravirtualization, and Hardware Assist, VMware White Paper
[7] http://en.wikipedia.org/wiki/Xen
[8] How does XEN work, December 2009, XEN
[9] http://en.wikipedia.org/wiki/VMware
[10] http://en.wikipedia.org/wiki/VMware_Workstation
[11] http://en.wikipedia.org/wiki/VMware_ESX
[12] VMware vSphere Online Library
[13] http://en.wikipedia.org/wiki/Hyper-V
[14] Hyper-V Architecture, Microsoft Developer Network Library
[15] System z PR/SM, IBM Systems Software Information Center
[16] Introduction to the New Mainframe: z/VM Basics, 2007, IBM Redbooks
[17] z/VM: General Information, 2009, IBM Redbooks

Heterogene Umgebungen von Mainframes
Seminar: Mainframes Today

Tobias Groll, Mathias Waldenburger
t_groll@cs.uni-kl.de, m_walden@cs.uni-kl.de

Database and Information Systems Group
University of Kaiserslautern
Supervisor: Karsten Schmidt

1 Einführung

Unter Mainframes versteht man IBM Großrechner der zSerie, die vor allem bei sehr kritischen Anwendungen ihren Einsatz finden.
Sie können mit sehr großen I/O-Raten umgehen und bieten Hochverfügbarkeit. Doch diese hohe Leistung hat auch ihren Preis, Mainframes sind sehr teuer in Anschaffung und Betrieb. Da es aus diesem Grund nicht rentabel ist, ausschließlich auf Mainframes zu setzen, gibt es noch eine ganze Reihe anderer Systeme, die in solchen Umgebungen verwendet werden. Dies können Standard x86 Rechner sein aber auch sehr spezialisierte Hardware wie der Smart Analytics Optimizer. Unter einer heterogenen Umgebung versteht man das Zusammenspiel verschiedener Komponenten, in diesem Fall die Komponenten um den Mainframe herum. Von den externen Maschinen können spezielle Serveraufgaben erledigt oder Infrastruktur verwaltet werden. Diese sind durch verschiedenste Anforderungen auch in jeglicher Zusammenstellung vorstellbar.
Im folgenden wollen wir Stellung nehmen zur Frage, welche Art von Anwendungen man auf dem Mainframe sinnvoll betreiben kann und welche besser auf den umgebenden Systemen Verwendung finden.

2 iSeries/AIX

2.1 Beschreibung

Die iSeries Server von IBM sind hochperformante Maschinen mit 1 bis 32 POWER Prozessoren mit 64 bis 512 GB Arbeitsspeicher. [3] Damit sind sie gerüstet für häufige Serveraufgaben wie Webhosting oder Terminaldienste. Auf den Servern laufen neben dem OS/400 System, welches auf diese Systeme zugeschnitten ist, auch AIX, Linux und Windows nativ sowie parallel in eigenen virtuellen Maschinen.
Im Gegensatz zu Mainframes sind sie nicht hochverfügbar und weniger gut gegen Ausfälle geschützt, weshalb sie nicht für sehr kritische Aufgaben geeignet sind. Außerdem hat ein einzelner Server eine deutlich geringere Leistung als ein Mainframe. Dennoch haben sie ihre Berechtigung, da nicht immer Hochverfügbarkeit

und hohe Leistung benötigt wird.
Bei den z Systemen wird mit sehr großem Aufwand Hochverfügbarkeit gewährleistet, ganze Rechenzentren sind redundant ausgelegt. Manche Anwendungen benötigen diese Ausfallsicherheit gar nicht und es ist somit erheblich günstiger, solche auf iSeries Maschinen auszuführen. Dort wird wesentlich weniger Aufwand betrieben, um eine hohe Verfügbarkeit bzw. Ausfallsicherheit zu ermöglichen.
Eine ganze Reihe von Applikationen werden speziell für OS/400 entwickelt. Falls diese Produktpalette nicht ausreicht, kann man zusätzlich auf die alternativen Betriebssysteme zurückreifen. So eignen sich die iSeries beispielsweise sehr gut als Webserver oder AIX Terminal Server, finden aber auch Verwendung als Datenbankserver oder SAP System mit geringer Benutzerzahl.

2.2 Einsatzbereich

AIX AIX ist ein Betriebssystem von IBM, welches der Unix-Spezifikation Version 3 entspricht. Es ist deshalb möglich, Unix-Programme in dieser Umgebung auszuführen. Außerdem verspricht IBM Linux-Affinität, was bedeutet, dass Linux Source Code voll mit AIX kompatibel ist und auch für dieses Betriebssystem kompiliert werden kann. [1] [10] Damit steht eine riesige Auswahl an Software zur Verfügung die diesem Betriebsystem vielfältige Nutzungsmöglicchkeiten für die iSeries Server bieten.
Das AIX Betriebssystem ist ausgelegt für Skalierbarkeit, Zuverlässigkeit und Administrierbarkeit. Ein mit AIX ausgestatteter Server kann dann beispielsweise als Terminal oder Webserver fungieren.

Terminal Server Oft werden die iSeries als AIX Terminal Server eingesetzt. Hierbei läuft auf einem zentralen Rechner das AIX System, auf dem man dann von Terminals direkt auf einer grafischen Oberfläche oder auch in der Konsole arbeiten kann. Wenn dieses System für kurze Zeit - zum Beispiel wegen eines Ausfalls - nicht in Betrieb ist, bedeutet das nicht gleich Millionenverluste für die Firma. Somit können normale Bürotätigkeiten durchgeführt werden, bei denen ein kurzzeitiger Ausfall keine gravierende Schäden anrichten kann.
Der Mainframe kann hierbei durchaus als Backend dienen, der Daten bereit stellt, verarbeitet und Transaktionen durchführt. Kritische Datenverarbeitung wird auf den Mainframe ausgelagert, weniger kritische, wie die Anzeige und Erfassung von Daten, kann direkt auf dem AIX System erfolgen. So werden die jeweiligen Stärken optimal ausgenutzt. Als Beispiel kann man hier eine Oberfläche zur Abwicklung von Bankgeschäften nennen, welche auf dem AIX-System des Terminal Servers läuft und zur Abwicklung der Transaktionen den Mainframe nutzt.

Webserver Ein Webserver hält Seiten und Applikationen bereit, die über das Netz aufrufbar sind. Diese Seiten können als Frontend für Anwendungen oder Webservices dienen. Der Webserver ist dann dafür zuständig, aus vorliegenden Daten Webseiten zu generieren und auszuliefern. Für die Datenverarbeitung

können andere Maschinen im Backend genutzt werden, falls diese Datenmenge sehr groß wird ist es oft sinnvoll, einen Mainframe zu nutzen. Dabei führt dieser Berechnungen durch oder wickelt Transaktionen ab, die über die Webseite angestoßen werden.
Für die Bereitstellung der Webseiten sind iSeries Server besser geeignet als der Mainframe selbst, da Webseiten auf Abruf bereitgestellt werden müssen und somit nicht ständig Last auftritt. Außerdem ist für das Anzeigen der Daten keine Hochverfügbarkeit notwendig, da hier bei einem Ausfall keine Daten verloren gehen. Der Mainframe im Backend genügt um ausreichende Sicherheit zu gewährleisten.
Für den Einsatz als Webserver dient Beispielsweise der Apache Webserver, welcher vollständig unter AIX betrieben werden kann oder aber auch zusammen mit einem Linuxsystem auf den iSeries Rechnern lauffähig ist. Der Apache ist die am weitesten verbreitete Software für Webserver. [11]

3 xSeries

3.1 Beschreibung

Die Intel Based Server von IBM sind Maschinen mit Intel Xeon Prozessoren. Sie verfügen über 4 bis 8 Prozessorkerne und 8 bis 100 GB Arbeitsspeicher. Kaufen kann man sie in normalen Desktop Gehäusen, aber auch als Blades für Serversysteme. Die Systeme sind im Vergleich sehr günstig, fast mit Desktop Rechnern vergleichbar. Sie sind einfach zu administrieren, da als Betriebssystem Windows oder Linux zum Einsatz kommen. Auch die einfachen Anwender PCs sind meist intel basierte Systeme und werden somit als intel based bezeichnet.

3.2 Einsatzbereich

Server Die intel based Server sind genau wie ihre großen Brüder, die pSeries, einsetzbar. Für Webseiten, die wenig Last aufweisen, optimal und immer noch schnell genug. Genauso eignen sie sich als Terminalserver für kleinere Benutzergruppen.

Terminals Standard Rechner, die auch als intel-based bezeichnet werden, können als Terminals dienen, mit denen man sich am Mainframe anmelden und auf diesem Arbeiten kann. So werden sie von den Systemadministratoren und Entwicklern, die an den z Systemen arbeiten, benutzt aber auch als Benutzerterminals von den Anwendern.
Außerdem ist es denkbar, sich über ein intel based Terminal an einem AIX Terminal Server anzumelden und an diesem zu arbeiten.

Storage-Mediatoren Ein Storage-Mediator verwaltet die Speichermedien. Er dient als Schnittstelle zwischen dem physisch Speicher und den Maschinen, die

diesen nutzen. Er bildet eine Abstraktionseben für den Speicherzugriff. Dadurch wird es ermöglicht, dass viele unterschiedliche Systeme auf viele verschiedene Speichermedien einheitlich zugreifen können.
Auch für diese Aufgabe kann ein xSeries Rechner verwendet werden. Ein Mainframe ist hierfür eher ungeeignet, da nicht soviel Rechenleistung für diese Aufgabe benötigt wird. Der Zugriff erfolgt on-demand, es wird also nicht immer die vollständige Leistung des Systems benötigt, da nicht ständig auf den Speicher zugegriffen wird und Speicherzugriffe auch gewisse Zeit dauern, in der die Rechenleistung des Systems kaum beansprucht wird. Hierfür wären die CPU Zeiten eines Mainframes einfach zu teuer.
Hier wird der xSeries Server zum Abstrahieren der Infrastruktur eingesetzt.

Control-Thinkpads Jeder Mainframe Server ist mit zwei Control-Thinkpads ausgestattet, welche direkt mit dem Mainframe verbunden sind. Von diesen Maschinen aus kann der Mainframe administriert werden, falls die Netzwerkverbindungen nicht mehr funktionieren. Außerdem kann der zSeries Server hierüber gebootet und heruntergefahren werden. Die Systemadministratoren können über diese Schnittstelle direkt am Mainframe arbeiten.

4 zBx - zEnterprise BladeCenter Extension

Die zEnterprise BladeCenter Extension ist eine Erweiterung für den Mainframe, welche als Host für pSeries und xSeries Blades dient. Sie ist direkt über ein privates Netzwerk mit dem Mainframe verbunden und wird auch von dessen IBM zEnterprise Unified Resource Manager verwaltet. So kann die Last optimal auf die gehosteten Servermaschinen verteilt werden. Die zBx bietet desweiteren Monitoring für ihre gehosteten Systeme, womit Last- und Leistungsdaten erfasst werden können.
Falls Anwendungen eingesetzt werden sollen, bei denen der Mainframe als Backend dient, Frontends aber auf iSeries oder intel-based Servern laufen sollen, können diese in der zBx gehostetet werden. So sind sie direkt mit dem Mainframe verbunden und werden nicht durch Übertragungsverzögerungen ausgebremst. Dies ist beispielsweise für Webserver relevant. Die Vorteile der Standard x86 Server werden so optimal in die Mainframeumgebung integriert. Auch der Smart Analytics Optimizer kann auf Hardware in einer zBx betrieben werden um ein optimales Zusammenspiel mit einer DB Umgebung auf dem Mainframe zu gewährleisten. [5]

5 Smart Analytics Optimizer

Der Smart Analytics Optimizer ist eine Erweiterung für den Mainframe, der Datenbankabfragen an ein installiertes DB2 System optimiert. Dafür analysiert er die Struktur der Abfragen und integriert die Erkenntnisse daraus in seine operationale Prozesse, zur Optimierung der Datenbank.

[4] Durch die immer größer werdenden Datenmengen kann die Verarbeitung von Anfragen an die Datenbank recht lange dauern. Die durchgeführten Optimierungen werden an die jeweilige Struktur angepasst und können Anfragen so sehr stark beschleunigen. Hierfür ist der Smart Analytics Optimizer zuständig. Er wird in einer zBx untergebracht, um direkt mit dem DB System auf dem Mainframe verbunden zu sein. Somit kann die Last gut verteilt werden. Rechenintensive Operationen werden beispielweise auf dem Accelerator ausgeführt.
Anfragen gehen weiterhin direkt an die Datenbank. Zur Optimierung werden sie dann an den Smart Analytics Optimizer weitergeleitet. Dieser analysiert die Anfragen sowie die Datenbank und stimmt seine Optimierungen auf die vorhanden Umgebung ab. Dafür indiziert er beispielsweise Spalten, auf denen oft Suchabfragen durchgeführt werden. Es muss kein manuelles Tuning der Datenbank mehr durchgeführt werden, da auch dies der Optimizer erledigt.
So können Datenbankanfragen um ein vielfaches beschleunigt werden sowie die Administration durch das Wegfallen von Tuningaufgaben vereinfacht werden. [7]

6 Migrations Konzepte

Häufig steht man vor der Frage, welches Konzept zum Einsatz kommen soll. Ist es sinnvoll, die verschiedenen Aufgaben alle auf einen Mainframe zu übertragen oder administriert man eine Vielzahl von kleineren Systemen, die jeweils verschiedene Aufgaben durchführen?
Vor diesem Problem stehen Großfirmen häufig und deshalb werden gelegentlich auch Systeme migriert. Zwei Beispiele sollen hier illustrien, dass beide Ansätze durchaus sinnvoll sein können.

6.1 Buisness Case

Migration auf den Mainframe Dass die Verwendung von Mainframes durchaus sinnvoll sein kann, zeigt das folgende Beispiel der Bank of Russia.
Diese betrieb 200 Server an 74 verschiedenen Standorten mit sechs Hardware Plattformen und drei unterschiedlichen Datenbanksystemen. Dies migrierten sie auf vier Mainframes in zwei Rechenzentren. Dadurch konnten sie die Kosten für eine Transaktion von 11 Rubel auf 50 Kopeken reduzieren. Dies wurde vorallem durch die Verringerung der Betriebskosten für die vielen Standorte erreicht. Beispielsweise sparten sie stark bei den Gehaltskosten, da sie das Personal in den Rechenzentren um mehr als 80% reduzierten.
An Software wird nach der Migration eine Oracle Datenbank auf z/OS eingesetzt sowie ein virtualisierter Linux Server auf z/VM, IBM WebSphere und IBM Tivoli für das Monitoring. Somit wurde die Softwarevielfalt stark verkleinert.
Ein weiterer Vorteil ist, dass sie nicht für Leistungsspitzen zusätliche Hardware kaufen müssen, sondern bei Bedarf Prozessoren des Mainframes zuschalten lassen können. Man bezahlt dann nur für die wirklich genutzte Rechenkapazität. [2]

Migration vom Mainframe Vielen Unternehmen ist die Unterhaltung von Mainframe - Systemen zu teuer und zu unflexibel. Deswegen hat HP sich über die Jahre Knowhow angeeignet um Neukunden zu gewinnen, indem sie deren Systeme vom Mainframe auf ihre HP Integrity Server migrieren können. Diese Flexibilität zeichnet sich durch einen annähernd linearen Zusammenhang zwischen Transaktionszahl und Serverkosten aus, das heißt bei Verdopplung der Transaktionen verdoppeln sich auch die Kosten.
Mainframes sind teurer bei Szenarien, bei denen eine bestimmte Zahl von Transaktionen nicht überschritten wird. [12] Die einzelne Transaktion wird auf den zSeries - Servern günstiger bei steigender Gesamtzahl.
Ein weiterer Aspekt ist die Softwarevielfalt. Führt man nur wenige lastintensive Anwendungen aus, eignet sich der Mainframe besser. Bei vielen verschiedenen Applikationen profitiert man jedoch von der Flexibilität vieler kleinerer Maschinen.
Hier sieht man, dass es auch möglich ist bei einer Migration vom Mainframe zu Serverfarmen Kosten einzusparen. Firmen wie Continental und Amadeus haben diesen Schritt mit Hilfe von HP bereits bewältigt. [9]

7 Green IT und Mainframe

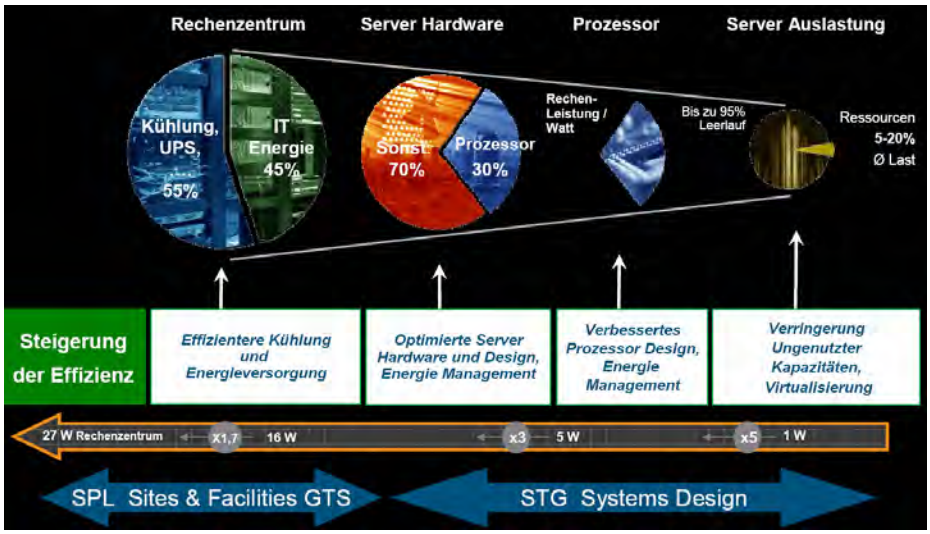


Abbildung 1. Energieverbrauch im Rechenzentrum [6]

Wie man auf Bild 1 sieht, setzt sich der Energieverbrauch eines modernen Rechenzentrums aus 55% Kühlung und 45% für die eigentliche Informationstechnik zusammen. Auf einem Mainframe können Systeme, die sonst auf vielen

Maschinen verteilt laufen, auf einer virtualisiert werden.
IBM fasste beispielsweise mehrere tausend Server auf nur 30 zSeries zusammen und erzielten somit einen um 80% niedrigeren Stromverbrauch. [8] Der Mainframe braucht zwar eine größere Kühlung als die einzelnen kleinen Maschinen, ein einzelner kann aber viele kleine Maschinen ersetzen.
Außerdem wird nach Bild 1 nur 30% der Energie, die der Rechner verbraucht, vom Prozessor benötigt, der Rest fällt auf die Perepherie, die beim Mainframe zentral vorhanden ist. Bei einem gut ausgelasteten Mainframe wird also weniger Energie benötigt als bei gleichwertigen Systemen mit vielen intel-based oder pSeries Servern. So kann man mit einem zSeries Server Energie einsparen.

8 Fazit

Alles in allem muss man sagen, dass sowohl Mainframes als auch andere Architekturen ihre Berechtigung haben und es immer auf die Anwendung ankommt, die auf dem jeweiligen System laufen soll. Mainframes haben große Vorteile in sicherheitskritischen Domänen. Hier sind beispielsweise Abwicklung von Transaktionen im Bankenwesen oder in Großfirmen zu nennen; Immer dann, wenn es essentiell wichtig ist, dass keine Daten verloren gehen oder Dienste ausfallen.
Im Bereich GreenIT hat der Mainframe Einsparpotential, da die Leistung vieler Rechner auf einer Plattform zusammengefasst wird und somit weniger Energie für die Kühlung benötigt wird.
Desweiteren zeichnen sich Server der zSerie durch hohe I/O Raten aus, dies ermöglicht erst die zeitgerechte Durchführung bei einer hohen Transaktionsdichte. Hierdurch wird auch Echtzeitfähigkeit, wie sie zum Beispiel bei Börsengeschäften notwendig ist, erreicht. Doch diese Vorteile haben auch ihren Preis. Mainframes sind nicht nur in der Anschaffung teuer sondern auch im Betrieb, was sich aus hohen Softwarelizenz- und Betriebskosten ergibt.
Für manche Anwendungsbereiche werden diese hohen Sicherheitsstandards und Datenraten gar nicht benötigt. Außerdem lohnt sich der Mainframe nur, wenn der Prozessor größtenteils ausgelastet ist. Manchmal ist es jedoch erforderlich, Leistungsreserven bereit zuhalten, wenn im Betrieb Leistungsspitzen zu bestimmten Zeiten zu erwarten sind. Dafür ist der Mainframe einfach zu teuer und normale x86 Server eigenen sich besser für diese Aufgaben. Hierzu zählen unter Anderem Webserver und Terminalserver.
Im Mainframeumfeld werden andere Maschinen auch für Verwaltung von Infrastruktur - wie Speicher und Netzwerk - eingesetzt. Auch Optimierungen der Abfragen wird von externer Hardware übernommen.
Um die Kommunikation zwischen Mainframe und anderen Rechnern zu optimieren, kann man diese auch in einer zEnterprise BladeCenter Extension unterbringen.
Wie man sieht ist es ratsam, die anfallenden Serverdienste genau zu analysieren und für die jeweilige Anwendung die richtigen Maschinen bereit zu stellen. Nur so kann man Kosten optimieren und das optimale Rechenzentrum für die eige-

nen Topics erhalten. Es wird deutlich, dass um den Mainframe herum zumeist noch eine ganze Reihe anderer Rechner zum Einsatz kommen.

Literatur

1. IBM. Ibm aix - unix on power systems including system p system i. http://www-03.ibm.com/systems/power/software/aix/about.html.
2. IBM. Bank of russia saves us$400 million per year by consolidating to ibm system z9. http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=PM&subtype=AB&appname=STGE_ZS_ZS_USEN&htmlfid=ZSC03037USEN&attachment=ZSC03037USEN.PDF, 2008.
3. IBM. Bladecenter blade servers. http://www-03.ibm.com/systems/bladecenter/hardware/servers/index.html, 2010.
4. IBM. Ibm smart analytics optimizer for db2 for z/os. http://www-01.ibm.com/software/data/infosphere/smart-analytics-optimizer-z/, 2011.
5. IBM. Ibm zenterprise system. http://www-03.ibm.com/systems/z/hardware/zenterprise/zbx.html, 2011.
6. D. S. Kammerer. Ibm zenterprise. Mainframe Summit Kaiserslautern, 2010.
7. J. Klockow. Ibm smart analytics optimizer. https://www-304.ibm.com/jct05001c/de/events/neue-aera-z/pdf/Klockow_System_z_Announement_ISAOPT.pdf, 2010.
8. T. Mach. Mainframes punkten bei energieverbrauch. http://www.computerwelt.at/detailArticle.asp?a=111702&n=22, 2007.
9. N.N. Auf hp integrity server migrieren – und die kosten um bis zu 70 prozent senken. http://h41131.www4.hp.com/at/de/press/mainframe-migration.html, 2008.
10. N.N. Aix. http://de.wikipedia.org/wiki/AIX, 2010.
11. N.N. Apache http server. http://de.wikipedia.org/wiki/Apache_HTTP_Server, 2010.
12. M. Schindler. 100 prozent richtig: Der mainframe ist teurer. aber.... http://www.silicon.de/management/cio/0,39044010,41001941,00/100_prozent_richtig_der_mainframe_ist_teurer__aber.htm, 2009.

An Overview over the basic z/OS Interfaces

Henrique Valer, Roxana Zapata
{hvaler,roxana.zogo}@gmail.com

Database and Information Systems Group
University of kaiserslautern
Supervisor: Dipl.-Inf. Karsten Schmidt

Abstract. This paper focuses on the Interactive Interfaces of z/OS and how users can interact with the system. Some components, like the TSO/E, ISPF and USS, are analyzed and compared based on their usability, accessibility and other main features.

Keywords: z/OS, TSO/E, ISPF, interfaces, usability, accessibility.

1 Introduction

The z/OS operating system is well known for its great features, concurrently user serving, reliable process of workloads, multiprocessing, multiprogramming, parallelism, and all of this so desired concepts that we hear all the time [10]. Such OS relies on heavy hardware, such as processors, hard disks (in IBM case, called DASD) provided by IBM "Big Irons", as mainframes are also known, achieving then also high availability and efficiency. But what would be of all this without a decent way of interacting with such structure? For that, the need of interfaces with high usability and accessibility becomes evident. Along with the development of the IBM operating systems, interfaces, commands and others ways of interaction with the system were created, and we are going to take a superficial look at some of them.

In section 2, we give a brief introduction about TSO/E and its facilities. Section 3 shows ISPF and its menu driven interaction system. Section 4 gives an outline over USS and its system interfaces, API and interactive z/OS shell interface. In section 5 some concepts about Batch Processing are clarified. In section 6 some security aspects are shown, such as identification and validation of z/OS users and finally in section 7 we present some of our conclusions.

2 TSO/E

Among other ways of interacting with the IBM z/OS operating system, one is known as a "native mode" and called TSO/E [6].

The Time Sharing Option/Extensions (TSO/E) provides basically a user login facility (because in z/OS each user obligatory has a user ID and a password)

and a command prompt interface, very similar to those used in DOS prompt [3] and Bash command line shells [5]. It also tries to increase usability and accessibility through two features, Information Center Facility and Connectivity Facility, respectively.

On its origin, as a component on the OS/390, it was composed of several commands and services designed for the end user [4]. Its main features includes, nowadays:

Information Center Facility Aiming to improve the mainframes usability, the main idea behind IC is to somehow facilitate the end-user live while struggling with the old fashioned mainframe interfaces. Through a series of panel driven menus (using ISPF technology, which is treated in section 3), the user can go through data, generate reports among other computational needs. An example of this interface can be seen in Fig. 1.

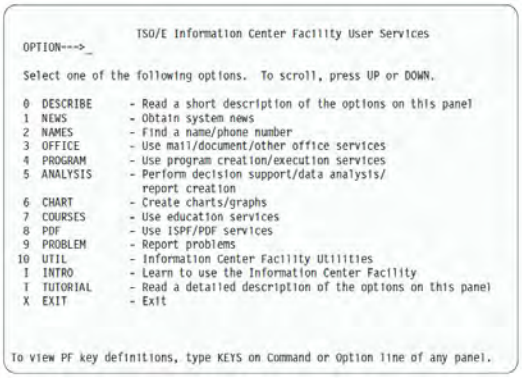


Fig. 1. Information Center Facility User Primary Panel[1]

Connectivity Facility The main idea is to allow a user to access an IBM System/370 through a normal PC, increasing the system accessibility. This upgrade on connectivity is achieved basically through a command called MVSSERV, which main purpose is to hold and manage the sessions on the host computer. After logging into TSO/E and starting MVSSERV, you can make requests from a IBM running Enhanced Connectivity programs. It consists basically of a router and an interface to the user, called Server-Requester Programming Interface (SRPI). This interface implements a CALL/RETURN mechanism and more details about it can be found in [7].

An overlook of such a mechanism can be seen in Fig. 2.

[1] Extracted from [4], Figure 2, page 12.

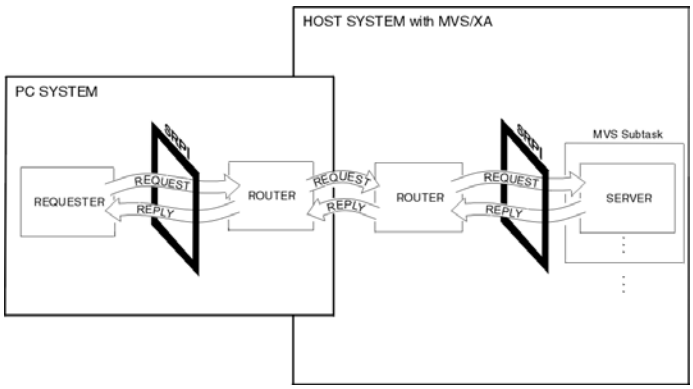


Fig. 2. The MVSSERV Enhanced Connectivity Environment[2]

3 ISPF

A very common way of interacting with z/OS is through ISPF instead of using the "native mode", TSO/E, explained in section 2. It is also considered an extension of this first mode, using the same connectivity features.

While its successor approach supports the interaction directly through commands, ISPF is basically a series of menus for interacting with the system. Aiming to increase usability, it includes an editor (on which data sets can be edited, commands can be entered, actions such as insert, copy, delete or move can be performed [10]), a browser, locating and listing facilities, among other functionalities. An idea of such interaction system can be seen in Fig. 3.

Besides that, after so many years using command line environments, the nature of computer systems evolved to graphical interfaces and ISPF somehow evolved together. There are several GUIs for ISPF, such as WSA, which allows the execution of ISPF in graphical mode (as seen in Fig. 4) and also file transfers between the mainframe and the PC station.

4 UNIX System Services

UNIX is a multi-task operating system whose first version was created in 1971[11]. Since that time, its use has been increasing year after year and nowadays UNIX based operating systems are quite common. UNIX offers a productive development environment, where multiple sets of tools and command languages exists. Such components are composable among themselves, which means their outputs can become inputs to other components, combining programs to perform complex tasks.

The z/OS provides a component called UNIX System Services (USS), that is basically an operating environment. It also provides two open systems interfaces [12] that allow access to z/OS UNIX.

[2] Extracted from [1], Figure 2, page 1.

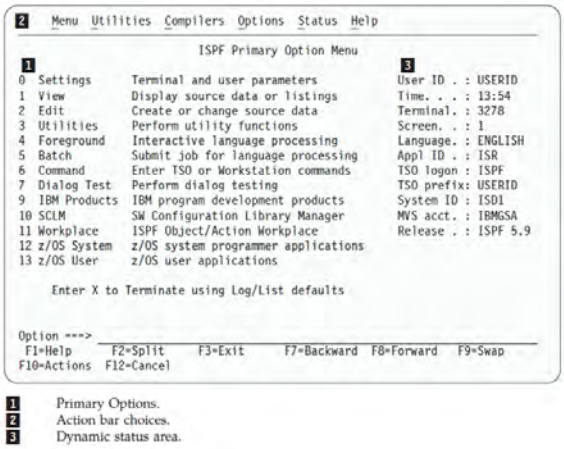


Fig. 3. ISPF Primary Option Menu[3]

Fig. 4. ISPF WSA GUI[4]

Application Program Interface: The main feature provided by this API is the debugger. An application programmer can use it to debug a z/OS UNIX System Services program written in C or C++. While some of these APIs are managed within the C Run-Time Library (RTL), the others access kernel interfaces to perform authorized system functions on behalf of unauthorized callers.

Interactive z/OS shell interface: One of the main components of this interface is the Shell, which can be used to enter shell commands, write shell scripts, and work with the file systems. These features are supported by the z/OS UNIX shell and the ISHELL interface. In addition, there are some TSO/E commands to support z/OS UNIX, but they have some knew limitations [13].

[3] Extracted from [6] Figure 4-8, page 159.
[4] Extracted from [25], Figure 13, page 34.

4.1 z/OS UNIX shell

Using the z/OS UNIX shell, a user interacts with UNIX-like interface (the shell). This command processor is based on the UNIX System V shell and has some features from the UNIX Korn Shell [14]. The shell commands behave similar to those coming from the UNIX System, but with the difference that the utilities on the z/OS UNIX shell are considered commands.

4.2 ISHELL

Instead of making your life a living hell, as its name proposes, this interface should help the user, using ISPF facilities. A user can perform many tasks through the ISPF dialogs using (ISHELL). There are two ways to invoke an ISPF panel: one way is from a TSO/E entering the command ISHELL, another way is from ISPF menu selecting the option ISPF shell. In Fig. 5, we can see the main panel of ISPF shell.

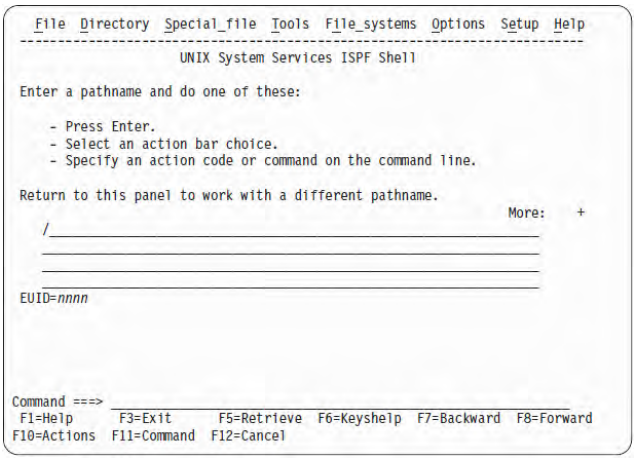


Fig. 5. Main panel of ISPF Shell [5]

Interactive z/OS users can choose between the methods mentioned above to invoke z/OS UNIX according to their background. Users familiarized with UNIX environment may use the UNIX shell, while those familiarized with TSO may use the command interface and the traditional panel interface.

5 Batch Processing

Batch processing is a mission-critical component that is used for almost all companies to handle processing of large data amounts. It is basically the execution

[5] Extracted from [11], Figure 19. Page 175.

of a program, possibly performed at scheduled time, with minimal human interaction. In the z/OS environment they are called batch jobs [21]. The batch processing elements are basically two: the Initiator and the Job itself.

The first element is an integral part of z/OS that reads, interprets, and executes the Job Control Language. It also manages the execution of batch jobs, one at a time, in the same address apace. JES does some JCL processing, but the initiator does the key JCL work. The second one is the job itself, represented mostly with Job Control Language and sometimes data intermixed with JCL.

Since batch processing aims for minimal human interaction and the whole processing occurs completely apart from the caller (who sent the job to be processed) why problems like usability and accessibility should be taken into account?

First of all, the whole usability issue goes into how easy is to develop batch jobs in JCL, and that is much beyond the scope of this work. Further references on how to develop JCL for everyday jobs and advanced skills can be found in [24].

Second, creating own communication mechanisms just to support the submission of batch jobs does not make much sense, in other words, why can not we use accessibility features existent in ISPF or TSO/E? Actually, we can, and that is not very complicated.

In the z/OS terminology, a job (work) is submitted for processing, where the job runs independently of the interactive session. The commands to commit a job might take any of the following forms:

- ISPF editor command line (SUBmit and press Enter)
- ISPF command shell (SUBmit USER.JCL where the data set is sequential)
- ISPF command line (SUBmit USER.JCL where the data set is sequential)
- TSO command line(SUBmit USER.JCL(MYJOB) where the data set is a library or partitioned data set containing member MYJOB)
- Use an enterprise scheduler, like Tivoli Workload Scheduler for z/OS [23]

This is, so to say, the client part, where jobs are submited. But what happens at the mainframe? Well, z/OS has two versions of the Job Entry Subsystem: JES2 and JES3 [22]. Their most basic functions are: accept jobs submitted in various ways, queue jobs waiting to be executed, queue jobs for an initiator, accept printed output from a job while it is running and queue the output, and send output to Output Manager.

The Fig. 6 shows three different points at which we can enter the SUBMIT command.

6 RACF

After talking so much about accessibility, another problem comes to mind: What if the mainframe is “too” accessible? What if the wrong persons have access to it? For that, security and authorization issues must be addressed.

[6] Extracted from [6], Example 6-5. Page 237.

Fig. 6. JCL processing[6]

The RACF [17] protects resources by denying unauthorized users to access protected resources. RACF keeps information about users, resources, and access authorities to further decide the permitted access, protecting the system resources.

The identification and validation of z/OS users is made through the RACF, giving a user ID and a system-encrypted password. When the RACF is inactive, the System Authorization Facility (SAF) security functions are established by default [18].

A user can perform security tasks using RACF commands from a terminal or workstation [19]. RACF commands enable to find out how users are defined in RACF, how to protect resources, how to change user's access to resources, and how to change the way RACF define access to these resources.

Instead of using RACF commands, a user can perform the security tasks mentioned above through RACF panels. To access the RACF panels, we need to choose the option R for RACF from the Interactive System Productivity Facility (ISFP), or use the ISPF interface from TSO/E mode. An example of this interface can be seen in Fig. 7

7 Conclusions

In this paper, we outlined the several ways of accessing and using z/OS operating system, thereby showing that users with different backgrounds can learn easily how to interact with z/OS.

The first options were TSO/E and an extension of it, the ISPF. While the first provided an Information Center Facility and a Connection Facility, improving usability and accessibility, the second uses the same connectivity features but

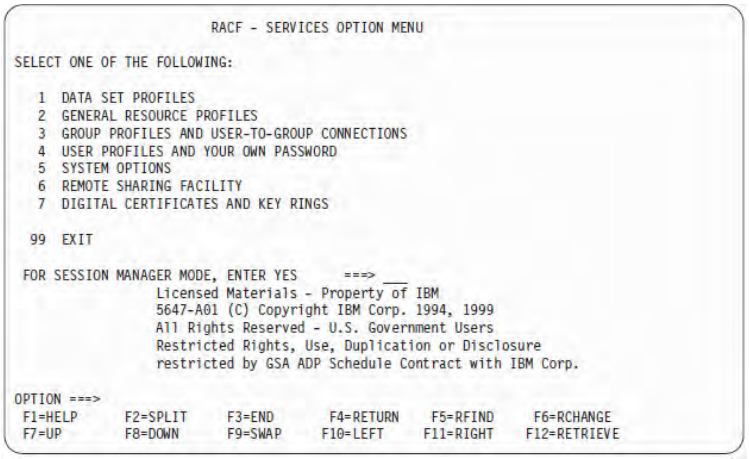


Fig. 7. The RACF primary menu panel [7]

has shown to be a better solution for the usability issues, through a series of interactive menus.

Besides that, a UNIX element called USS allows the use of UNIX in the z/OS. Great news for the UNIX users, who does not need to learn all the tricks of mainframes (just some of them) and are able to connect via rloging or telnet commands.

Of course all these interconnection facilities increase security issues, that are addressed with RACF. With a secure environment, the process of batch jobs becomes more reliable.

We can conclude that, in one hand the connectivity and accessibility problems seams to have quite good solutions, there are several ways of connecting with such operating system and they are quite secure. In the other hand, usability in such environments is still a problem, but z/OS has been trying to increase usability over time, always providing new features.

[7] Extracted from [17], Figure 1. Page 5.

Glossary

API Application Programming Interface. 1, 4

DASD Direct Access Storage Devices. 1
DOS Disk Operating System. 2

GUI Graphical User Interface. 3, 4

IBM International Business Machines. 1, 2, 10
IC Information Center. 2
ISHELL ISPF Shell. 4, 5
ISPF Interactive System Productivity Facility. 1–7, 10

JCL Job Control Language. 6, 7, 10
JES Job Entry Subsystem. 6

MVS Multiple Virtual Storage. 10

OS Operating System. 1
OS/390 An earlier version of the z/OS operating system . 2, 10

PC Personal Computer. 2, 3

RACF Resource Access Control Facility. 7, 8, 10
RTL Run-Time Library. 4

SAF System Authorization Facility. 7
SRPI Server-Requester Programming Interface. 2

TSO/E Time Sharing Option/Extension. 1–7, 10

USS UNIX System Services. 1, 3, 8

WSA Work Station Agent. 3, 4

z/OS Operating system for IBM mainframe computers. 1, 3–8, 10

References

1. z/OS V1R7.0-V1R11.0 TSO/E Guide to Server-Requester Programming Interface, SA22-7785
2. z/OS TSO/E Command Reference, SA22-7782
3. Adams, M.: MS-DOS: an introduction. Century Communications (1985)
4. z/OS TSO/E General Information, SA22-7784
5. Newham, C.; Rosenblatt, B.: Learning the bash Shell, O'Reilly Media, Inc. (2005)
6. Ebbers, M.; Kettner, J.; O'Brien, W.; Ogden, B.; Ayyar, B.; Duhamel, M.; Fremstad, P.; Martinez Fuentes, L.; Gelinski, M.; Grossmann, M.; Hernandez, O.; Yuiti Hiratzuka, R.; Mller, G.; Neufeld, R.; Newton, P.; Seubert, B.; Thorsen, H.; R. Wilkinson, A.: Introduction to the new z/OS mainframe, IBM Redbook (2009)
7. z/OS V1R7.0-V1R9.0 Using REXX and z/OS UNIX System Services, SA22-7806
8. z/OS V1R11.0 ISPF Dialog Developer's Guide and Reference, SC34-4821
9. z/OS V1R11.0 ISPF User's Guide Vol II, SC34-4823
10. z/OS Concepts, http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zconcepts/toc.htm
11. z/OS V1R1.0 UNIX System Services User's Guide, SA22-7801
12. Rogers, P.; Robert Hering, P.; Bruinsma, P.; Antoff, T.; O'Connor, N.; Khner, L.; Sousa, L.: UNIX System Services z/OS Version 1 Release 7 Implementation, IBM Redbook (2006)
13. z/OS V1R11.0 UNIX System Services Command Reference, SA22-7802
14. UNIX on OS/390: A powerful blend, http://www-03.ibm.com/systems/z/os/zos/features/unix/release/nmas.html
15. z/OS V1R11.0 UNIX System Services Planning, GA22-7800
16. Rogers, P.; Salla, A.; Sousa, L.: ABCs of z/OS System Programming Volume 10, IBM Redbook, (2008)
17. Security Services RACF General User's Guide, SA22-7685; Sixth Edition, September 2009.
18. Weller, R.; Clements, R.; Dugdale, K.; Fremstad, P.; Hernandez, O.; C Johnston, W.; Kappeler, P.; Kochersberger, L.; Tedla, A.; Thompson, J.; Venkatraman, A.: Introduction to the New Mainframe: Security, IBM Redbook, (2007)
19. z/OS Security Server RACF V1R6.0 Command Language Reference, SA22-7687
20. The single UNIX Specification, version 2 on UNIX98, Open Group, White Paper.
21. Batch Modernization on z/OS, SG24-7779
22. z/OS MVS JCL Users Guide, SA22-7598
23. Tivoli Workload Scheduler, http://www-01.ibm.com/software/tivoli/products/scheduler/
24. Menendez, R.; Lowe, D.: Murach's OS/390 and z/OS JCL, Mike Murach and Associates, 2002
25. z/OS V1R9.0 ISPF User's Guide Vol I, SC34-4822

Jobmanagement in z/OS

Seminar: Mainframes Today

Max Bechtold, Martin Hiller

Technische Universität Kaiserslautern
Lehrgebiet Informationssysteme

1 Einleitung

Im Mainframe Summit 2010 wurden Aspekte von z-Systemen aus vielen verschiedenen Blickrichtungen thematisiert. Dabei wurde deutlich, dass der Hauptteil der Arbeit auf einem z-System in Form sogenannter Batchaufträge oder *Jobs* erledigt wird. Deren Definition, Verwaltung und Ausführung soll in dieser Arbeit zusammenfassend vorgestellt werden.

Zunächst wird hierbei auf Batchverarbeitung im Allgemeinen eingegangen, die sie in z-Systemen steuernden Komponenten und wie diese interagieren. In einem zweiten Teil ist die Definition von Jobs in Form von Skripten Untersuchungsgegenstand.

2 Konzepte der Batchverarbeitung

Batchverarbeitung ist ein integraler Bestandteil von z-Systemen. Zunächst soll diese Form der Verarbeitung kurz charakterisiert werden, bevor die Umsetzung in z/OS und die beteiligten Komponenten vorgestellt werden.

2.1 Charakterisierung

Die programmgestützte Verarbeitung von Daten kann generell in zwei Arten klassifiziert werden. *Interaktive Verarbeitung* geschieht ad hoc, unter direkter Beteiligung des Benutzers, der ein Computersystem verwendet. Daraus folgt, dass das System nur sehr beschränkte Möglichkeiten hat, den Auftrag mit anderen zu koordinieren und in einen Plan einzureihen, der eine optimale Auslastung ermöglicht. Wenn der Benutzer einen Job absetzt, muss er direkt ausgeführt werden.

Im Gegensatz dazu werden Jobs in der *Batchverarbeitung* nur geplant und zur Ausführung in Auftrag gegeben. Das System erhält zumeist einen zeitlichen Rahmen, in dem der Job abzuarbeiten ist, kann den Ausführungszeitpunkt innerhalb dessen jedoch frei planen. Damit kann eine bessere Auslastung eines Systems erreicht werden; zudem können aufwändige, aber nicht dringende Aufgaben z. B. zu Nachtzeiten, wenn Systemlasten typischerweise eher niedrig sind, durchgeführt werden. Damit belegen sie keine Ressourcen, die für Programme

in der interaktiven Verarbeitung benötigt werden. Im Gegensatz zu diesen sind die Benutzer auch nicht an der Ausführungsphase beteiligt.

Während im *Personal-Computer*-Bereich interaktive Verarbeitung vorherrscht, wird bei z/OS die meiste Arbeit in Form von Batchjobs erledigt. Darum enthält dieses Betriebssystem auch einige zentrale Komponenten zur Unterstützung von Batchverarbeitung. Diese und deren Interaktion werden im Folgenden erläutert.

2.2 Komponenten

In diesem Abschnitt sollen an der Verarbeitung von Jobs beteiligte Komponenten und Systeme kurz vorgestellt werden. Diese werden in Abb. 1 in Zusammenhang gesetzt.

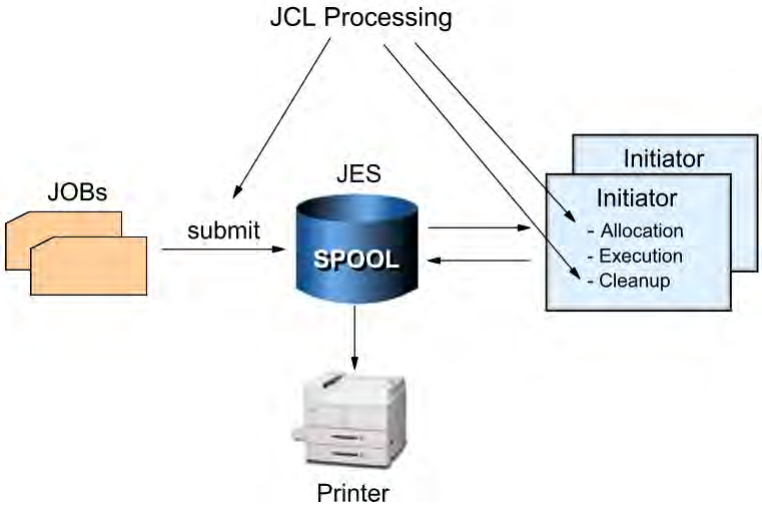


Abbildung 1. Zusammenspiel der Komponenten der Batchverarbeitung (Quelle: [1])

Job Entry Subsystem (JES) Die Batchverarbeitung auf einem z-System wird nicht direkt vom Betriebssystem z/OS kontrolliert, sondern vom *Job Entry Subsystem*, kurz JES. Diese Subkomponente nimmt Batchaufträge entgegen und plant deren Ausführung. Dabei können Prioritäten für die in Warteschlangen verwalteten Jobs angegeben werden. Die beiden verfügbaren Versionen dieser Komponente, JES2 und JES3, werden zunächst kurz gegenübergestellt.

JES2 und JES3 ermöglichen beide das Einstellen, Verwalten und Ausführen von Batchaufträgen; in Details unterscheiden sie sich jedoch. Bei Einzelprozessorsystemen mit JES3 kann der Benutzer Bänder, Abhängigkeiten zwischen und

Deadlines für Jobs über das Job Entry Subsystem erledigen, während JES2 diese Möglichkeit nicht bietet. Hier müssen diese Einstellungen anderweitig erledigt werden. Noch deutlicher ist der Unterschied in Systemen mit mehreren Prozessoren, auf denen je eine JES2-Instanz läuft. Jede dieser verwaltet ihre Jobeingabe, -planung und -ausführung selbst und unabhängig von anderen JES2-Instanzen. In JES3 existiert dagegen ein einzelner, globaler JES3-Prozessor, der diese Aufgaben der Batchverarbeitung koordiniert und alle JES3-Instanzen aufeinander abstimmt.

In Sysplex-Systemen, bei denen mehrere z-Systeme mit dem Ziel höherer Redundanz zusammengeschaltet sind, kann eine JES2-Instanz Spool Data Sets und andere Daten mit weiteren JES2-Systemen desselben Clusters teilen. Dies bezeichnet man als *Multi-Access Spool* (MAS). Auch hier ist JES3 umfassender ausgelegt, da es nun die komplette Jobverwaltung nicht nur zwischen einzelnen Prozessoren, sondern zwischen verschiedenen Systemen eines Sysplex koordiniert.

Während bei JES2 Geräteallokation nur vom z/OS-Basiskontrollprogramm durchgeführt werden kann, kann dies bei JES3 auch über das Job Entry Subsystem vorgenommen werden.

Da auf z-Systemen vorrangig JES2 verwendet wird, beziehen sich die Erläuterungen im Folgenden auf diese Version.

Spool Während der Verarbeitung verwaltet JES2 die Ausgaben, z. B. Logs, die durch einen Job anfallen. Dazu benutzt es ein virtuelles Disk Data Set, den *Spool*, der physisch durch ein oder mehrere Data Sets repräsentiert wird. Die Technik, Ein- und Ausgabedaten von Jobs in die Spool Data Sets einzuspeisen, nennt sich *Spooling*. Auch das Zwischenspeichern von Daten, die zu einem späteren Zeitpunkt weiterverarbeitet werden, oder die Übergabe von Daten an einen Drucker, die gedruckt werden sollen, während parallel weitergearbeitet werden kann, fällt unter diesen Begriff.

Initiatoren Jobs werden durch das Job Entry Subsystem allerdings nur angestoßen, die eigentliche Durchführung übernehmen *Initiatoren*. Jeder Initiator wird in seinem eigenen Adressraum des Systems gestartet und kann einen neuen Job zur Ausführung anfordern, woraufhin JES2 einen Auftrag aus einer entsprechenden Warteschlange entnimmt und dem Initiator übergibt. Dieser kümmert sich dann um die Allokierung von Geräten, löst Referenzen auf Programme auf, die in der JCL eines Jobs verwendete werden, und stellt sicher, dass bei der Ausführung mehrere Jobs zur selben Zeit keine Konflikte bei Zugriff auf dieselben Data Sets auftreten. Die dazu notwendigen Informationen bezieht der Initiator aus den Befehlen im JCL-Script eines Jobs. Nachdem ein Batchauftrag beendet wurde, erledigt der Initiator Aufräumarbeiten. Die Anzahl aktiver Initiatoren bestimmt den Grad an Nebenläufigkeit von Batchaufträgen.

Es gibt zwei Arten von Initiatoren: JES2- und WLM-Initiatoren. Erstere werden beim Systemstart initialisiert und einer Jobklasse zugewiesen. Den JES2-Initiatoren werden von JES2 nur Jobs ihrer Klasse zugeteilt, natürlich unter Be-

achtung der Priorität bzw. der Warteschlangenposition. WLM-Initiatoren werden automatisch vom z/OS-*Workload-Manager* gestartet, je nach Auslastung bzw. freien Systemressourcen. Die optimale Ausnutzung letzterer ist wesentliche Aufgabe des Workload-Managers, dem im Fall von WLM-Initiatoren auch die direkte Auswahl und Planung von Jobs unterliegt.

2.3 Ablauf

Nun sollen die verschiedenen Phasen, aus denen die Abarbeitung eines Batchauftrags besteht, beschrieben werden. Zwischen diesen Phasen befinden sich jeweils Warteschlangen, die einen asynchronen und zeitverzögerten Übergang erlauben.

Eingabephase: In der Eingabephase nimmt JES2 Jobs entgegen. Dies kann über Eingabeströme, Eingabegeräte oder durch andere Programme geschehen. Für letzteres werden sogenannte *interne Reader* als Schnittstelle verwendet, über die auch Jobs selbst weitere Jobs an JES2 übergeben können. JES kann mehrere interne Reader besitzen, über die Aufträge auch zeitgleich empfangen werden können. Ist eine JES-Instanz Teil eines Job-Entry-Netzwerks, kann es auch Aufträge von anderen Knoten dieses Netzwerks annehmen.

Nachdem ein Auftrag angenommen wurde, werden dessen JCL und benötigte Eingabedaten in Spool Data Sets geschrieben. Wenn ein angenommener und geplanter Auftrag zur Ausführung ausgewählt wird, liest JES2 die Jobdaten vom Spool, die dann in den folgenden Phasen verwendet werden.

Konvertierungsphase: Bevor ein JCL-Script ausgeführt werden kann, muss es zunächst konvertiert werden. Dies erfolgt mit Hilfe eines Konvertierungsprogramms, das Referenzen auf System- und Benutzerprozedurbibliotheken auflöst und damit die JCL ergänzt. Dieses neue Skript wird anschließend in maschinenlesbaren *interpreter text* umgewandelt, mit dem sowohl JES2 als auch Initiatoren umgehen können.

Auch der so übersetzte Code wird auf Spool Data Sets zwischengespeichert. Wurden bei der Konvertierung keine Fehler im JCL-Script entdeckt, wird der Job in die Warteschlange der Verarbeitungsphase eingereiht. Ansonsten werden entsprechende Fehlermeldungen in Fehler-Data-Sets vermerkt.

Verarbeitungsphase: Während dieser Phase reagiert JES2 auf Initiatoren, die gerade neu gestartet wurden oder einen Job abgearbeitet haben, und die einen neuen Job anfordern. JES2 entnimmt einen Auftrag aus einer Warteschlange und teilt diesen dem Initiator zu. Im Fall von JES2-Initiatoren muss zusätzlich deren Jobklasse berücksichtigt werden.

Ausgabephase: In der Ausgabephase bearbeitet JES2 die Ausgabe, die von einem Job produziert wird. Dazu gehören auszudruckende Systemnachrichten oder Data Sets. Diese werden u. A. nach ihrer Ausgabeklasse gruppiert und in

die Warteschlange für die Ausgabe in Form von Ausdrucken bzw. selten auch Lochkarten gesetzt.

Druckphase: Die Ausgabe durch Ausdrucke oder Lochkarten erfolgt nach den Ausgabeklassen. Sie kann entweder lokal oder entfernt ausgeführt werden. Nachdem dieser Schritt beendet wurde, wird der Job zur Entfernung in die Aufräumwarteschlange eingereiht.

Aufräumphase: Am Ende der Verarbeitung eines Batchauftrags steht die Aufräumphase. Hier wird der Speicherplatz in Spool Data Sets, der für diesen Auftrag verwendet wurde, wieder freigegeben, der somit weiteren Jobs zur Verfügung steht. Abschließend wird eine Nachricht an den Operator abgesetzt, dass der Job beendet wurde.

In Abb. werden diese Phasen und ihre Übergänge in einer Übersichtsgrafik dargestellt.

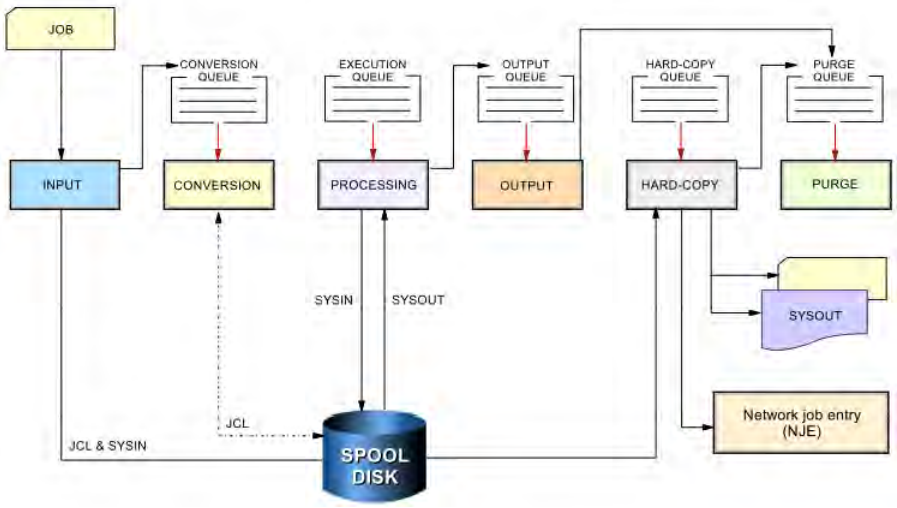


Abbildung 2. Phasen der Batchverarbeitung (Quelle: [1])

3 Definition und Verwaltung von Jobs

In diesem Abschnitt wird im Wesentlichen die sogenannte *Job Control Language* (kurz: JCL) thematisiert, die zur Definition von (Batch-)Jobs in z/OS genutzt wird. Dazu wird im Folgenden die grundlegende Syntax von JCL anhand konkreter Anwendungsbeispiele erläutert und anschließend noch einige fortgeschrittene Nutzungsmöglichkeiten dieser Sprache aufgezeigt. Am Ende folgt eine kurze Vorstellung der *System Display and Search Facility* (SDSF) als wichtigstes Werkzeug zur Verwaltung von Jobs und zur Anzeige der produzierten Ausgabe.

3.1 Job Control Language (JCL)

JCL ist eine Sprache zur Definition von Jobs. Im Wesentlichen beschreibt ein JCL-Script, welches Programm (geschrieben in COBOL, Assembler oder PL/I) oder welche Folge von Programmen mit welchen Parametern auszuführen ist. Die fertige JCL wird im Anschluss an das System (oder genauer: dem *Job Entry Subsystem*) übergeben, welches die JCL-Definition interpretiert, den Job ausführt und Ausgaben erzeugt. Dieser grundlegende Ablauf wird in Abb. 3 dargestellt.

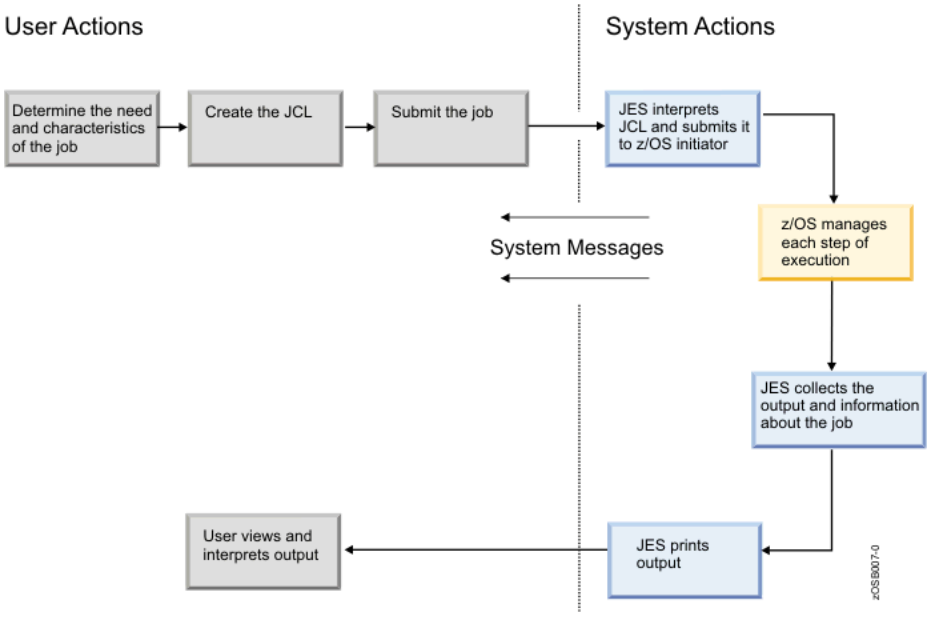


Abbildung 3. Verwendung von JCL zur Jobdefinition (Quelle: [1])

Ein Job, wie er in JCL spezifiziert wird, besteht aus einem oder mehreren Schritten, die jeweils einem Programmaufruf entsprechen und sequentiell abgearbeitet werden. Zu jedem Schritt sind wiederum Datendefinitionen anzugeben,

die beschreiben, woher das jeweilige Programm seine Eingabedaten erhält und wohin eventuelle Ausgaben zu schreiben sind. Natürlich können bei Angabe mehrerer Schritte die Ausgaben eines Schrittes von nachfolgenden Schritten verwendet werden. Die eben beschriebene Hierarchie (Job → Schritt → Datendefinition) spiegelt sich auch in der JCL-Syntax wider. So besteht eine Jobdefinition aus den drei grundlegenden Befehlen JOB (zur Angabe allgemeiner Jobinformationen), EXEC (entspricht einem Schritt bzw. Programmaufruf) und DD (für Datendefinitionen). Als Beispiel einer Jobdefinition soll folgende (vereinfachte!) JCL dienen:

```
//MYJOB   JOB  (123456)
//MYSTEP  EXEC PGM=PROGRAMNAME
//INFILE  DD   DSN=USER.INFILE,DISP=SHR
//OUTFILE DD   DSN=USER.OUTFILE,
//             DISP=(NEW,CATLG,DELETE)
/*
```

Bei diesem konstruierten Beispiel handelt es sich um einen Job mit dem namen „MYJOB“, der als einzigen Schritt das Programm mit dem Namen „PROGRAMNAME“ ausführt. Dieses Programm erwartet die zwei Parameter „INFILE“ und „OUTFILE“, die in der Jobdefinition auf konkrete Data Sets gesetzt werden. Dabei wird auf das erste Data Set („USER.INFILE“) lesend zugegriffen, während das zweite Data Set („USER.OUTFILE“) im Zuge der Ausführung neu erzeugt wird und zur Speicherung der Programmausgabe dient.

Wie im Beispiel zu erkennen ist, besteht jede Befehlszeile in JCL aus folgenden Elementen:

Doppelslash (//): Die führenden *Slashes* müssen zu Beginn (Spalte 1 und 2) jedes Befehls gesetzt werden.

Name: Name des jeweiligen Jobs/Schritts/Datendefinition (bis zu 8 Zeichen lang)

Operation: Name der durchzuführenden Operation. Darunter fallen die drei Statements JOB, EXEC und DD.

Operand: Jede Operation erwartet zusätzliche Operanden (Parameter), die über „Operandname=wert“ gesetzt werden. Die Angabe der Operanden muss in Spalte 16 der Zeile beginnen. Mehrere Operanden werden durch Kommata getrennt und können über mehrere Zeilen definiert werden. Wichtig dabei ist das abermalige Setzen des Doppelslashs zu Beginn der Zeile.

Die Definitions des Jobs endet mit der Zeichenfolge der letzten Zeile (/*). Als Nächstes sollen die drei Operationen JOB, EXEC und DD inklusive ihrer Operanden näher betrachtet werden.

JOB-Statement: Über die Operanden des JOB-Statements werden allgemeine Angaben zum Job gemacht. Ein ausführlicheres Statement als im obigen Beispiel könnte wie folgt aussehen:

```
//MYJOB JOB   (123456),'Max Muster',CLASS=A,PRTY=6,
//            MSGCLASS=1,MSGLEVEL=(1,1),NOTIFY=&SYSUID
```

Beim ersten Parameter handelt es sich um eine *Accounting* Nummer, gefolgt von dem Namen des Autors dieser JCL. Da Jobs in verschiedene Klassen eingeteilt werden können, ist die Angabe des Operanden CLASS möglich, dessen Wert aus einem Zeichen (Buchstabe von A bis Z oder Ziffer von 0 bis 9) bestehen muss. Der Parameter PRTY (für *priority*) gibt die Priorität des Jobs innerhalb der jeweiligen Klasse an, wobei 0 die niedrigste und 15 die höchste Priorität ist. Die beiden Operanden MSGCLASS und MSGLEVEL geben an, welche System- und JCL-Nachrichten (→ MSGLEVEL) auf welchen Ausgabegeräten (→ MSGCLASS) anzuzeigen sind. Über den NOTIFY-Operanden kann ein Benutzer angegeben werden, der über den Erfolg oder Misserfolg der Jobverarbeitung informiert wird. Im obigen Beispiel wird derjenige Benutzer informiert, der den Job aufgegeben hat.

EXEC-Statement: Das EXEC-Statement definiert einen Schritt des jeweiligen Jobs und dient im Wesentlichen der Angabe des auszuführenden Programms. Dazu wird im Operand PGM (für *program*) der Name des Programms genannt. Zwar hat auch die EXEC-Operation weitere optionale Operanden, dessen Erklärung im Zuge dieser Einführung zu weit gehen würde.

DD-Statement: Jedes DD-Statement ist eindeutig mit einem Schritt assoziiert und gibt eine Datenquelle (z.B. ein Data Set, aber auch andere E/A-Geräte) an, die das Programm zur Ausführung benötigt. Der DD-Name zu Beginn der Befehlszeile ist beim DD-Statement nicht frei wählbar, sondern muss dem entsprechenden DD-Namen im Programm gleichen, damit die Datenquelle richtig gebunden werden kann. Wissen über das auszuführende Programm und seine Schnittstelle ist also in jedem Fall notwendig. Die zu einem Schritt angegebenen Datendefinitionen sind nicht zwangsläufig eingehende Daten, sondern können auch vom Programm zur Ausgabe von Ergebnissen genutzt werden.

Wie im ersten Beispiel gezeigt, dient der Operand DSN (für *Data Set Name*) zur Angabe eines Data Sets als Datenquelle für das Programm. Je nach Nutzung des Data Sets durch das Programm sind weitere Angaben zum verwendeten Data Set notwendig. Wird auf das Data Set nur lesend zugegriffen oder benötigt das Programm exklusiven Zugriff? Existiert das angegebene Data Set bereits oder muss es zunächst erzeugt werden? Diese und weitere Einstellungen werden von dem Operanden DISP (für *disposition*) gesteuert. DISP ist ein Tripel, das aus folgenden Werten besteht:

Status: gibt den Status des angegebenen Data Sets an. Dabei können folgende Werte gesetzt werden:
NEW: Data Set existiert nicht und soll neu angelegt werden.
OLD: Data Set existiert bereits.
MOD: Data Set wird neu erzeugt, falls es noch nicht existiert.

SHR: Data Set existiert bereits, der Zugriff auf das Data Set erfolgt lesend ($\rightarrow SHR = shared$).

Normal Disposition: Mit diesem Parameter wird angegeben, was mit dem spezifizierten Data Set nach erfolgreichem Abschluss des entsprechenden Schrittes geschehen soll.

Abnormal Disposition: Analog beschreibt dieser Parameter, was im Fehlerfall mit dem angegebenen Data Set zu tun ist.

Für die beiden Parameter *Normal* bzw. *Abnormal Disposition* sind folgende Werte möglich:

DELETE: Das Data Set wird gelöscht.

CATLG: Das Data Set wird in den Systemkatalog aufgenommen.

UNCATLG: Das Data Set wird aus dem Systemkatalog entfernt.

KEEP: Das Data Set wird behalten ($\rightarrow$ keine Aktion durchzuführen).

PASS: (nur für *Normal Disposition*) Das Data Set wird an nachfolgende Schritte des aktuellen Jobs weitergereicht.

Die im Beispiel verwendete Einstellung für das Ausgabe-DS

```
DISP=(NEW,CATLG,DELETE)
```

legt also das Data Set neu an, katalogisiert es bei erfolgreicher Ausführung und löscht es im Fehlerfall. Für den Operanden DISP ist es außerdem möglich, nur den ersten Wert (z.B. DISP=SHR) oder nur die ersten beiden Werte (z.B. DISP=(NEW,CATLG)) des Tripels anzugeben.

Für das DD-Statement sind außer dem DSN- und dem DISP-Operanden noch weitere Angaben möglich (teilweise notwendig), die sich unter anderem auf die physischen Position des Data Sets (z.B. welches Volume?) und Details der Allokation (z.B. primäre und sekundäre Größe) beziehen.

Nach der Erklärung der grundlegenden Syntax und Funktionsweise von JCL wird nun auf einige erweiterte Funktionen dieser Sprache eingegangen.

Konkatenation von Data Sets: Data Sets desselben Typs (z.B. *partitioned Data Sets*) können konkateniert und dem Programm als *eine* Datenquelle übergeben werden:

```
//MYJOB  JOB   (123456)
//STEP   EXEC  PGM=PROGRAM
//INFILE DD    DSN=TEST.FILE1
//             DSN=TEST.FILE2
//             DSN=TEST.FILE3
/*
```

Die Datenquelle SYSIN: Das Übertragen von Daten bzw. Parameter an das auszuführende Programm muss nicht zwingend über Data Sets erfolgen. Falls das Programm es unterstützt, können Daten auch direkt in der JCL angegeben werden. Dazu wird eine DD mit dem Namen SYSIN spezifiziert und im Anschluss (Text-)Daten mitgegeben.

```
//MYJOB  JOB   (123456)
//STEP   EXEC  PGM=PROGRAM
//SYSIN  DD    *
827456
/*
```

Instream Procedures: JCL erlaubt die Definition und Benutzung von Prozeduren, die im Grunde als parametrische Jobs zu verstehen sind. Genau wie ein Job definiert eine Prozedur einen oder mehrere Schritte, die sequentiell auszuführen sind. In Prozeduren werden allerdings noch nicht sämtliche Operanden mit konkreten Werten belegt oder alle benötigten DDs spezifiziert. Die Angabe dieser verbleibender „Parameter“ erfolgt erst bei Aufruf der Prozedur. *Instream Procedures* werden im Gegensatz zu *Cataloged Procedures* direkt in der jeweiligen JCL definiert. Dazu werden die Operationen PROC (Beginn der Prozedur) und PEND (Ende der Prozedur) verwendet. Dazwischen werden sämtliche (parametrische) Schritte angegeben, die im Zuge der Prozedur auszuführen sind. Aufgerufen werden Prozeduren ähnlich wie Programme durch das EXEC-Statement, wobei die Parameter der Prozedur im Anschluss belegt werden müssen.

```
//MYJOB   JOB  (123456)
//*-----------------------------------*
//MYPROC  PROC
//STEP    EXEC PGM=PROGRAM
//INFILE  DD   DSN=&INPUT,DISP=SHR
//OUTFILE DD   DSN=USER.OUTFILE,
//             DISP=(NEW,CATLG,DELETE)
//MYPROC  PEND
//*-----------------------------------*
//MYSTEP  EXEC MYPROC,INPUT=USER.INFILE
/*
```

Ebenso ist das Überschreiben bzw. Ergänzen von DDs möglich. Dazu werden nach dem EXEC-Statement des aufrufenden Schrittes (im Beispiel: MYSTEP) weitere DDs angegeben, die die gleichnamigen DDs in der Prozedurendefinition überschreiben.

Zeitschranken: Zu jedem Job bzw. jedem Schritt kann über den TIME-Operanden eine obere Grenze für die Ausführungszeit angegeben werden. Wird eine dieser Zeitschranken (ob nun eine Job- oder eine Schritt-Zeitschranke) bei der Ausführung überschritten, bricht der Job automatisch ab.

Bedingte Ausführung: Falls ein Job aus mehreren Schritten besteht, kann die Ausführung eines Schrittes an den *Return Code* eines vorangegangenen Schrittes geknüpft werden. Dazu wird als zusätzlicher Operand im EXEC-Statement eine Bedingung angegeben, die bei Erfüllung den aktuellen Schritt überspringen lässt.

```
//MYJOB  JOB   (123456)
//STEP1  EXEC  PGM=PROG1
//STEP2  EXEC  PGM=COBPROG,
//             COND=(8,EQ,STEP1)
/*
```

Im Beispiel wird STEP2 übersprungen, falls der *Return Code* von STEP1 den Wert 8 liefert ($\rightarrow EQ = equals$).

IF-THEN-ELSE Konstrukt: Mächtiger als die bedingte Ausführung von Schritten ist das IF-THEN-ELSE Konstrukt. Nach der Ausführung eines Schrittes kann hiermit eine Bedingung geprüft und je nach Erfüllung verschiedene nachfolgende Schritte durchgeführt werden.

```
//MYJOB   JOB  (123456)
//STEP1   EXEC PGM=PROGRAM1
//COND    IF STEP1.RC = 0 THEN
//STEP2   EXEC PGM=PROGRAM2
//CONDE   ELSE
//STEP3   EXEC PGM=PROGRAM3
//        ENDIF
/*
```

Neben dem *Return Code* (RC) können weitere Eigenschaften eines Schrittes in der IF-Bedingung geprüft werden (z.B. Prüfung auf abnormales Ende).

3.2 System Display and Search Facility (SDSF)

Die *System Display and Search Facility* (SDSF) ist ein Tool zur Überwachung, Kontrolle und Anzeige von Jobs und der erzeugten Ausgabe. Aus dem Hauptmenü dieses Programms (siehe Abb. 4) lässt sich dessen Funktionalität erahnen.

Für gewöhnlich wird SDSF nach dem Einreichen eines Jobs verwendet, um bei erfolgreicher Ausführung die Ausgabe des Jobs zu betrachten oder ggf. JCL-Fehler zu korrigeren. SDSF bietet allerdings noch einige zusätzliche Funktionen. Zu den wichtigsten Funktionen gehören:

- Betrachten des System-Logs
- Eingabe von Systemkommandos
- Überwachung und Kontrolle laufender Jobs
  (z.B. Pausieren oder Abbrechen von Jobs)
- Anzeige von Job-Ausgaben
- Kontrolle über die Reihenfolge, in der Jobs ausgeführt werden
  (bzw. in welcher Reihenfolge die Ausgabe gedruckt wird)
- Kontrolle über Drucker und Initiatoren

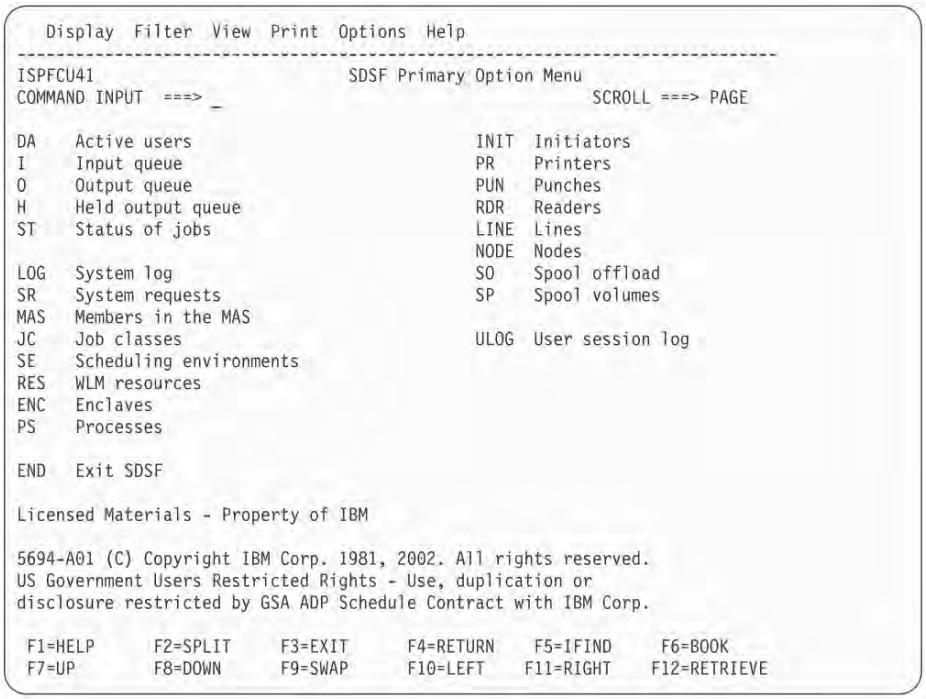


Abbildung 4. SDSF Hauptmenü (Quelle: [2, S.96])

4 Fazit

In dieser Arbeit wurden wichtige Konzepte und Komponenten der Batchverarbeitung in z/OS zusammengefasst. Während das Job Entry Subsystem Jobs entgegennimmt und zur Ausführung auswählt und vorbereitet, übernehmen Initiatoren die eigentliche Verarbeitung. Der Spool dient dabei als Ein- und Ausgabegerät sowie als Zwischenspeicher.

Im zweiten Teil wurde JCL als Skriptsprache für Jobs vorgestellt und auf die Einteilung in Schritte, typische Bestandteile wie Jobname, DD- und EXEC-Statements und weitere Sprachmittel zum Setzen von Zeitschranken und Verwenden von Prozeduren eingegangen. Zudem wurde SDSF als zentrales Werkzeug zur Batchverarbeitung charakterisiert.

Glossar

DD	Data Definition; gibt eine Datenquelle zur Ausführung eines Programms an	7
DISP	Disposition; Operand im DD-Statement, der den Zugriff auf den angegebenen Data Set genauer spezifiziert	8
DSN	Data Set Name; Operand im DD-Statement	8
EXEC	Execute-Statement; entspricht einem Programmaufruf in einem Job	7
JCL	Kurz für *Job Control Language*, Sprache zur Definition von Jobs	6
JES	Kurz für *Job Entry Subsystem*; z/OS-Komponente, die die Verarbeitung von Jobs kontrolliert	2
SDSF	System Display and Search Facility; Tool zur Überwachung, Kontrolle und Anzeige von Jobs und Job-Ausgaben	11
Spool	Kurz für *simultaneous peripheral operations online*; virtuelles Data Set zum Empfangen/Übergeben von Ein- und Ausgabedaten von Jobs	3, 13
Spooling	Verwenden eines Spools	3

Literatur

1. IBM Corporation: z/OS basic skills information center - z/OS concepts. `http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp`, [Last Access: 01.12.2010]
2. IBM Corporation: z/OS basic skills information center - z/OS concepts. `http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zconcepts/zconcepts_book.pdf`, [Last Access: 16.12.2010]
3. IBM Corporation: Batch processing and the Job Entry Subsystem (JES). In: Introduction to the new mainframe (2006)
4. IBM Corporation: Using Job Control Language (JCL) and System Display and Search Facility (SDSF). In: Introduction to the new mainframe (2006)
5. Ramesh Krishna Reddy: JCL Study Material. `http://www.mainframegurukul.com/srcsinc/drona/programming/languages/jcl/jcl.html`, [Last Access: 01.12.2010]
6. Wikipedia: Job Control Language. `http://de.wikipedia.org/wiki/Job_Control_Language`, [Last Access: 01.12.2010]

Programming languages on z/OS
Seminar: Mainframes Today

Simon Eberz, Thomas Eickhoff
s_eberz@cs.uni-kl.de, t_eickho@cs.uni-kl.de

Database and Information Systems Group
University of Kaiserslautern
Supervisor: Karsten Schmidt

Abstract: There are a number of different programming languages on z/OS. Several types of programming languages (compiled, interpreted, scripting) will be contrasted and their respective features, characteristics and fields of application will be discussed. Furthermore, we will highlight the differences between desktop and mainframe software development. Finally, we will provide a short tutorial to illustrate the programming workflow on the mainframe.

1 Introduction

"FORTRAN's tragic fate has been its wide acceptance, mentally chaining thousands and thousands of programmers to our past mistakes."
– Edsger W. Dijkstra

While the first mainframes were programmed using punch cards this has changed with the introduction of assembly language. While this development provided some comfort to programmers (e.g. automatic increase of the program counter), assembler programs were still hard to write and debug. Furthermore, the assembler programs were not portable at all, the code was written for one specific architecture. This obviously resulted in a low efficiency as the code had to be rewritten in order to be used with a different architecture (e.g. due to a new processor). Therefore, so-called "higher" programming languages were required. The development of these languages started with the release of FORTRAN in 1955. FORTRAN was developed by IBM in order to provide a solution to the biggest problems caused by assembler. In order to achieve this goal, FORTRAN introduced several new concepts (control structures like loops, conditional execution of code ...) to provide a higher level of abstraction. Of course, the evolution didn't stop with FORTRAN, over the time newer programming languages like C++ and Java found their way to the mainframes and brought innovative concepts like object oriented programming with them. This term paper will provide an overview of the development and usage of several languages and will contrast their individual characteristics and fields of use.

2 Classification

"Suppose you went back to Ada Lovelace and asked her the difference between a script and a program. She'd probably look at you funny, then say something like: Well, a script is what you give the actors, but a program is what you give the audience. That Ada was one sharp lady..."
– Larry Wall

There are several ways to classify the numerous programming languages according to implementation details, supported paradigms or areas of application. As was mentioned in the introduction, the most obvious scale is the provided level of abstraction from the hardware:

Low level languages are conceptually close to the hardware, allowing the programmer to access specific features of the underlying architecture. On the other hand, they are more difficult to use, as they provide very few "comfort features".
High level languages provide higher levels of abstraction, making the programs more portable as well as easier to write and maintain, at the cost of being less performant and hiding some hardware-specific details and functions.

These terms do not provide an absolute classification into two groups of languages, as there are many different levels of abstraction which have been introduced in even more programming languages. A Java programmer might consider C's memory management "low level" while an assembler programmer probably would not see C as particularly close to the hardware.

A less ambiguous separation of programming languages is the difference between compiled and interpreted languages.

Code of **compiled** languages is translated into machine language before the resulting program is run directly on the hardware.
Interpreted programs are translated "line by line" by an interpreter while they are executed.

However, there are languages which can be both compiled and interpreted (e.g. Common Lisp, Haskell) as well as languages which are compiled to an intermediate code (e.g. Java Bytecode, Microsoft CIL).

Another group of programming languages are the so-called **scripting languages** (e.g. REXX, JavaScript, Perl). This term connotates a high level, usually interpreted language, which is not powerful or performant enough to write large stand-alone programs. Their high level of abstraction makes them suitable for smaller tasks and the management of simple processes (e.g. shell scripting).
The separation between a "scripting language" and a "programming language" has become almost meaningless in the last years, as the progress in computing power made it possible for scripting languages to be used in larger programs despite their limited performance (e.g. Ruby on Rails).

3 Software Development on z/OS

"I had a running compiler and nobody would touch it. ... they carefully told me, computers could only do arithmetic; they could not do programs."
– Grace Hopper

There are several characteristics which distinguish software development on z/OS from "standard" software development.The most significant difference is that the development platform is usually a PC, while the target platform is a mainframe. This leads to the programmer not being able to test his or her code on the development platform without the use of an emulator. Additionally, the programmer might not have access to the target platform all the time, as mainframe computing time is expensive. Furthermore, it is never a good idea to run experimental or potentially malicious code on a production system. The price for mainframe systems and the inability of some firms to afford such a system for testing purposes only makes special emulator software necessary.

Depending on the platform being used at a given point of time, the applications that are used for development are different. Whereas programmers tend to use TSO/ISPF while on the mainframe, more familiar IDEs (IBM's Rational IDE is based on Eclipse) are used while on a desktop computer. Rational also includes a terminal emulator for using ISPF on a remote mainframe. Mainframe programmers also

need to think about their program's availability requirements, which includes careful error handling as well as timing constraints which are heavily influenced by the program's workload and the number of users using the application at a given time. In conjunction with these constraints, it has to be decided whether the application will be run in batch mode or online, for example any application that requires online user interaction should be run online whereas processes which mainly manipulate tape drive data should be run in batch mode. In both cases, the application can be run within a CICS region.

Another aspect that makes writing code for z/OS different is the alternative file system using data sets and members instead of files and folders (for an example, see section **6.2.1**). The programmer also needs to specify how and where the data will be stored and accessed, which requires a broad knowledge of the storage devices (e.g. DASD, tape drive), access methods (e.g. VSAM) and/or database management systems (e.g. DB2). At an even lower level, there are differences in the formatting of data, the most obvious being the using of the EBCDIC rather than ASCII as character encoding. The particular steps from source code to a compiled program are similar to the ones found in desktop programming. The source code is compiled, linked, bound, loaded and executed. An additional phase of precompilation is however needed, if the program issues EXEC CICS commands or EXEC SQL calls. In this step, these calls are translated to COBOL, PL/I or assembler, depending on the source language in question.

4 IBM Basic Assembly Language

"The effective exploitation of his powers of abstraction must be regarded as one of the most vital activities of a competent programmer."
– Edsger W. Dijkstra

Basic assembly language (also known as HLASM or High Level Assembler) is the assembly language used to program IBM Mainframe computers. As mentioned before, assembly language is very close to the computers internal functions, which makes assembler code both extremely efficient and extremely difficult to write, since one instruction of a higher level programming language often translates to several assembler instructions (such as **l**oad or **s**hift **r**ight **l**ogical).

First Appearance	1964
Developed By	IBM
TIOBE Index	$< 0.25\%$
Key Features	
Paradigm	N/A
Fields of Application	Performance-critical applications

Table 1: IBM HLASM

For programmer convenience, HLASM allows the use of macros, which are parametrized text replacements. Those can be used to simulate procedures (or, more precisely, inline procedures).

5 Compiled Languages

z/OS supports a number of compiled languages. The characteristics as well as the specific areas of application will be described in the following subsections.

5.1 COBOL

"The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense."
– Edsger W. Dijkstra

The COmmon Business-Oriented Language (COBOL) was designed by Grace Hopper in 1959, making it one of the oldest programming languages still used today. COBOL was first standardized in 1968, the latest standard being COBOL 2002, which introduced object oriented programming, XML generation and several other "modern" features.

First Appearance	1959
Developed By	Grace Hopper aka "Grandma COBOL"
TIOBE Index	0.405%
Key Features	"English-like syntax"
Paradigm	Procedural, Object Oriented (since 2002)
Fields of Application	Business data processing, Compatibility, legacy code

Table 2: COBOL

COBOL's feature set caters to the requirements found in business programming. For example, COBOL uses decimal arithmetic by default (floating point numbers were only added in the 2002 standard). Variables in COBOL can be formatted by using a picture clause, which defines the structure of all values that can be stored in the variable, e.g. `PIC 999` for an unsigned 3-(decimal-)digit number, `PIC S999` for a signed 3-digit number or `PIC A(8)` for 8 alphabetic characters (including space). Using `PIC 99` to declare the storage for a year was the source of many Y2K-related bugs.

Additionally, COBOL uses an "English-like" syntax, which makes it supposedly easier to read or understand "for non-programmers such as managers, supervisors and users". For example, the C statement `a = b + c;` is equivalent to the following COBOL code: `ADD b TO c GIVING a.`

These features make COBOL suitable for business computing, while its old age and the accordingly large COBOL code base necessitate the ongoing use of the language on the Mainframe for backwards compatibility.

5.2 PL/I

"When FORTRAN has been called an infantile disorder, full PL/1, with its growth characteristics of a dangerous tumor, could turn out to be a fatal disease."
– Edsger W. Dijkstra

PL/I was designed as a programming language for both scientific and business computations (in order to be used in the application domains of both FORTRAN and COBOL). As such, PL/I incorporated features from both these languages (e.g. the picture clauses for both arithmetic and character types) as well as Algol's block semantics, a combination of features which made the language both difficult to learn and hard to implement. Because of this, the language never became the successor to neither FORTRAN nor COBOL and remains a minority on the Mainframe.

PL/Is syntax is somewhat less verbose or "English-like" than COBOL's, in allowing abbreviations for some of the languages keywords (e.g. DECL for DECLARE, PROC for PROCEDURE). Today's programmers might also find the syntax for arithmetic expressions and assignment (`foo = baz + bar;`) more familiar than their COBOL counterparts.

First Appearance	1964
Developed By	IBM
TIOBE Index	$< 0.25\%$
Key Features	
Paradigm	Procedural
Fields of Application	Business data processing

Table 3: PL/I

5.3 Java

"Java is, in many ways, C++--. "
–Michael Feldman

Java was developed by Sun Microsystems and released in 1995. It is a object-oriented language with great influences from C and C++, with which it shares a big part of its syntax. Java source code is compiled to byte code and can afterwards be run on every Java Virtual Machine. This "write once, run everywhere" feature of Java enables programmers to easily run Java programs on mainframe computers, even if they had not specifically written for this platform. In practice, however, this is not especially advised, as mainframe software usually has requirements which greatly differs from those for "standard" software. Nevertheless, Java's platform-independence has probably contributed to its wide usage (as indicated by its TIOBE Index). Java, being a general-purpose programming language, is used for development of web- as well as desktop-applications.

z/OS Java programs provide the same API as standard Java programs, but also incorporate several changes regarding the special file system of z/OS.[APPLZ] Another change, though mostly invisible to the programmer, is the fact, that Java programs are often run on specialized zSeries® Application Assist Processors (zAAP) [ZAAP]. These processors are stripped-down versions of standard processors, their benefit lies not so much in gained efficiency, but in reduced licensing costs for the software. Additionally, the use of these specialized processors obviously reduce

First Appearance	1995
Developed By	Sun Microsystems
TIOBE Index	18.509%
Key Features	
Paradigm	Object-Oriented
Fields of Application	Web applications, General purpose

Table 4: Java

the load of general-purpose processors. However, these benefits correspond to the number of Java programs used in the corresponding environment.

6 Interpreted Languages

There are several interpreted languages available on z/OS. While the borders between compiled and interpreted languages have blurred, there are still some special characteristics which will be described below.

6.1 CLIST & JCL

CLIST ("Command LIST"[3]) is a relatively simple interpreted language developed by IBM for MVS and TSO scripting. As the name suggests, CLIST programs are lists of OS commands combined with control structures (if-then-else, loop), comparable to UNIX shell scripts. CLIST has now been largely replaced by REXX. It is, however, still used as "glue language" combining the different commands of a JCL batch process. Note however, that any scripting language could be used; CLIST is mainly used for backwards compatibility. JCL (the Job Control Language) alone is not sufficient for scripting, as on z/OS it only allows three statements:

First Appearance	N/A
Developed By	IBM
TIOBE Index	$< 0.25\%$
Key Features	
Paradigm	N/A
Fields of Application	Scripting

Table 5: CLIST

- JOB statements containing meta-information such as the job's title or billing information,
- EXEC statements identifying the program and how it should be run and
- DD statements containing Data Definitions, i.e. which data will be used by the process.

This limited syntax originates from the early versions of JCL where the descriptions were submitted using punch cards.

6.2 REXX

REXX (the REstructured eXtended eXecutor) was developed by IBM in order to provide a simple, high level language to their customers. REXX is now the most used scripting language on z/OS, having replaced EXEC, EXEC2 and the majority of CLIST, except for legacy code. REXX has a simple syntax including conditionals, looping constructs and procedures. REXX includes some features typically found in scripting languages, e.g. the simple type system (everything is stored as a character string) and the ability to interpret strings as REXX code (using the INTERPRET statement). In the following section, we show how a simple REXX script is written and executed.

6.2.1 "99 Bottles Of Beer On The Wall" – A Tutorial

In this section we want to demonstrate the steps necessary to run a simple REXX script on z/OS. First, we must allocate a data set and create a new member for our script.

First Appearance	1979
Developed By	IBM
TIOBE Index	$< 0.25\%$
Key Features	
Paradigm	Procedural, Object oriented (ObjectRexx)
Fields of Application	Scripting

Table 6: REXX

- After calling up the ISPF Primary Option Menu, choose 3 (confirm each selection by pressing Return) to enter the Utility Selection Panel.
- Here, choose 2 ("Data Set"). Enter "OURPROJ" as Project, "REXX" as Group and "EXEC" as Type. Finally enter "a" into the ISPF command line ("Option ===>") and confirm to allocate the data set.
- If you do not want to mess around with the settings, press Return, F3, F3 to return to the Primary Option Menu.
- Choose 2 ("Edit"). In the Editor Entry Panel, enter "BOTTLES" as the new members name and confirm.

Now it's time to enter the actual script. Copy the following code into the editor (Adapted from [BOTTL]).

```
/* REXX VERSION OF THE BOTTLES PROGRAM */
/* ORIGINAL VERSION BY R. MARK ENRIGHT */
DO I = 99 TO 1 BY -1
    IF I > 1 THEN PLURAL_1 = 'BOTTLES'
        ELSE PLURAL_1 = 'BOTTLE'
    IF I = 2 THEN PLURAL_2 = 'BOTTLE'
        ELSE PLURAL_2 = 'BOTTLES'
    SAY I PLURAL_1 'OF BEER ON THE WALL,' I PLURAL_1 'OF BEER'
    SAY 'TAKE ONE DOWN, PASS IT AROUND,' I-1 PLURAL_2 'OF BEER ON THE WALL.'
END
SAY 'NO MORE BOTTLES OF BEER ON THE WALL, NO MORE BOTTLES OF BEER'
SAY 'GO TO THE STORE AND BUY SOME MORE'
EXIT
```

After you have entered the contents of the script (by the way, as REXX scripts have to start with a comment, it would not be wise to skip the first two lines), press F3 twice. Back in the Primary Option Menu, you can run your script by entering "TSO EXEC REXX.EXEC(BOTTLES) EXEC" into the command line.

7 CICS

CICS is IBM's transaction server used primarily on z/OS and z/VSE, although various ports exist for other IBM OSes (OS/2 and i5/OS) as well as AIX, Windows, Solaris and HP-UX (under the name TXSeries). The first version was published in 1969. CICS manages several regions, which contain one "Quasi-Reentrant Task Control Block" (QR TCB) and possibly other tasks/TCBs. A region might be spread across several z/OS systems. CICS region can be started both interactively or as a batch process. A CICS will then manage the atomic execution of the specified QR TCB. Other tasks can be run, for example, to access an external database. CICS transactions can be started using EXEC CICS statements (as mentioned in section **3**).

8 Conclusion

"About the use of language: it is impossible to sharpen a pencil with a blunt axe. It is equally vain to try to do it with ten blunt axes instead."
– Edsger W. Dijkstra

The previous sections illustrated the different requirements which arise from software development for mainframe computers. The greatest discrepancy lies in the software life-cycle. While standard desktop applications are usually replaced after a few years, mainframe software is used much longer. This is a consequence of the usage of some of the software for management of long-term contracts like life insurances. Obviously, a company can't afford of not being able to use or maintain a certain program due to lack of support for the corresponding programming language. This directly justifies the continued use of very old languages like COBOL for maintenance of this "legacy code". Nevertheless, the use of "modern" programming languages must be possible as well, as completely new programs should not be written using a programming language which has been outdated for several decades. In order to accommodate these differences, the mainframe architecture has to be as versatile as possible. There is a fundamental trade-off between the required amount of backwards compatibility and a desirable amount of performance and maintainability. Finally, a programmer must always keep in mind, that he writes programs, which most likely are crucial for a company's success, and thus have much higher requirements for robustness and reliability.

When choosing a programming language for a new project, there are several important factors one has to consider. While it might be useful to stick to one single language in order to improve interoperability between distinct programs, this might lead to unmaintainable programs, as in the future no one might be able to efficiently understand these languages. For example, today it is much harder to find COBOL developers, than a person who has a good knowledge of Java. It might even be beneficial to use several languages for one single project, given the characteristics of each languages as described in the previous sections.

9 Sources

[APPLZ] z/OS Basic Skills Information Center Application Programming on z/OS
http://publib.boulder.ibm.com/infocenter/zos/basics/topic/com.ibm.zos.zappldev/zappldev_book.pdf

[TIOBE] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[ZAAP] http://www-03.ibm.com/systems/z/hardware/features/zaap/index.htm

[BOTTL] http://99-bottles-of-beer.net/language-rexx-1894.html

[TBPL] Peter A. Henning, Holger Vogelsang: Taschenbuch Programmiersprachen. (2007)

... as well as the slides from the KL IBM Mainframe Summit 2010

Working with z/OS: Applications
Seminar: Mainframes Today

Peter Binnig, Juri Krieger
p_binnig@cs.uni-kl.de, j_kriege@cs.uni-kl.de

Database and Information Systems Group
University of Kaiserslautern
Supervisor: Karsten Schmidt

1 Einleitung

Im Jahre 1991, sagte der damalige Herausgeber der Infoworld Stewart Alsop

> *"I predict that the last mainframe will be unplugged on 15 March 1996"*[3]

heute wissen wir, dass diese Vorhersage falsch war und noch immer Mainframes im Betrieb sind und weitere Mainframes in Betrieb genommen werden. Das liegt vor allem an der überholung der Mainframe Hardware und der Investition in Software. Beides führt dazu, dass Unternehmen weiterhin ihren leistungsstarken und zuverlässigen *"Big Iron"* verwenden können und der Zugang zu neuen Technologien, wie Java, Linux oder auch das Internet, nicht verschlossen bleiben. Ohne seine Anwendungen würde der Mainframe jetzt nicht da stehen wo er ist, sondern wäre, wie Stewart Alsop prophezeit hat, abgeschaltet worden.

Die Bedeutung, eines Mainframes für Unternehmen, wird wohl am besten beschrieben mit:

> *An obsolete device used by thousands of obsolete companies serving billions of obsolete customers and making huge obsolete profit for their obsolete shareholders.*
>
> *And this years models run twice as fast as last years for half of the prize.*[2]

2 z/OS

z/OS ist ein viel genutztes Betriebssystem von IBM für dessen Mainframes, das eine stabile, sichere, dauerhaft verfügbare und skalierbare Umgebung für die darauf arbeitenden Anwendungen bietet. Dabei ist z/OS eine *share-everything* Laufzeitumgebung, die mittels spezieller Hardware und Software sicherstellt, dass eine hohe Ausnutzung der Komponenten gewährleistet wird und Systemressourcen bestmöglich ausgeschöpft werden.

Wird z/OS von einem Kunden bei IBM bestellt, wird der Systemcode über das Internet oder auf physischen Magnetbändern geliefert. Der Systemcode wird

von einem z/OS Systemprogrammierer auf Externspeicher (*Direct Access Storage Devices*, kurz **DASD**) kopiert, angepasst und mit hilfe der Systemkonsole gestartet. Während z/OS ausgeführt wird, befindet es sich im Hauptspeicher und teilt sich diesen mit allen anderen Anwendungen, die auf dem Mainframe ausgeführt werden. Da die Mainframes dafür entwickelt worden sind um große Datenmengen zu verarbeiten und tausende von Benutzern gleichzeitig zu bedienen, unterstützt z/OS *multiprogramming* (Multitasking) und *multiprocessing* (Mehrprozessorsysteme). z/OS benötigt für diese Aufgaben große Mengen an Hauptspeicher und komplexe Sicherheitsmechanismen um die geforderte Verfügbar, Sicherheit und Skalierbarkeit des Systems zu gewährleisten. Dazu stellt z/OS Systemkomponenten wie den *Workload Manager* (**WLM**) oder den *Recovery Termination Manager* (**RTM**) bereit. Systemfunktionen, die von Benutzern oder Anwendungen, häufig genutzt werden, wie z.B. das öffnen von Dateien, können schnell mit hilfe von ausführbaren Macros aufgerufen werden.

Um die Kommunikation innerhalb von z/OS und den darauf ausgeführten Anwendungen zu gewährleisten, stellt z/OS eine Vielzahl von *control-blocks* (Kontrollblöcken) zur Verfügung. Diese können jeweils in eine der folgenden 4 Kategorien zugeordnet werden:

- *system control block*s repräsentieren ein z/OS System und beinhalten unter anderem Informationen darüber wie viele Prozessoren verwendet werden
- *resource control block*s beinhalten Informationen zu jeweils einer einzigen Ressource (u.a. Prozessor, DASD)
- *job control block*s repräsentieren einen laufenden Job
- *task control block*s repräsentieren eine Arbeitseinheit

Ein häufig verwendeter control block ist der *service request block* (**SRB**), welcher zum Anfordern von Systemdiensten verwendet wird. Dabei wird der SRB vom SCHEDULE Makro als Input verwendet, welcher für die Einplanung asynchroner Routinen zuständig ist.

z/OS verwendet *real storage* (Hauptspeicher) und *auxiliary storage* (DASD's) für *virtual storage*. Jeder Benutzer und jedes Programm bekommt von z/OS seinen eigenen *address space* (Adressraum) zugewiesen, welcher aus einer Sequenz von virtuellen Adressen, beginnend bei 0, besteht. z/OS muss für diese virtuellen Adressen ein mapping auf reale Adressen durchführen. Dieser Prozess wird *Dynamic Address Translation* (**DAT**) genannt. Diese Adressräume bestehen aus privaten und öffentlichen Bereichen. Um die Abwärtskompatibilität zu gewährleisten besitzen Adressräume zwei private Adressbereiche, einen für 24-bit Adressierungen unterhalb der 16 Megabyte Linie ("The Line") und einen weiteren privaten Adressbereich oberhalb der 16 Megabyte Linie für 31-bit Adressierungen. Der restliche Adressbereich ist ein öffentlicher Adressbereich auf den Programme und Benutzer anderer Adressräume zugreifen können. Fehler die im privaten Adressbereich stattfinden betreffen ausschließlich den eigenen Adressbereich, wodurch die Zuverlässigkeit des Systems erhöht und Recovery Maßnahmen erleichtert werden. Für die Kommunikation zwischen den isolierten Adressräumen gibt es in z/OS zwei Möglichkeiten, entweder es wird ein SRB an den scheduler geschickt (asynchron) oder es werden *cross-memory services*

benutzt um auf die Register zuzugreifen (synchron). Für die cross-memory services werden spezielle Rechte benötigt, die von der *Authorized Program Facility* (**APE**) kontrolliert werden. *Cross-memory* (**XM**) ist dabei eine Entwicklung die aus dem *virtual storage* hervorging und verfolgt dabei drei Ziele:

- Daten synchron zwischen zwei virtuellen Adressen in unterschiedlichen Adressräumen zu verschieben
- Abgeben der Kontrolle an Instruktionen die sich in einem anderen Adressbereich finden
- Ausführen einer Instruktion in einem Adressbereich, während ihre Operanden sich in einem anderen Adressbereich befinden

Da *auxiliary storage* und *real storage* verwendet werden um die Illusion des *virtual storage* zu kreiren, werden verschiedene Speicherstrukturen für das jeweilige Speichermedium benötigt. *Frames* werden auf dem real storage gespeichert, *pages* ist die Speicherstruktur die vom *virtual storage* verwendet wird und die Daten die auf dem *auxiliary storage* gespeichert werden, befinden sich in *slots*. Wie bereits erwähnt befinden sich nur die aktiven Teile eines Programmes im *real storage* und die inaktiven Teile werden auf die DASDs ausgelagert. Dieser Prozess wird *paging* genannt. Um das paging zu vereinfachen haben alle 3 Speicherstrukturen eine Größe von 4 KiB.

Falls ein Programm mit höherer Priorität läuft und kein *real storage* mehr zum einlagern (*paging in*) zur Verfügung steht, werden Programme mit geringerer Priorität von z/OS in den *auxiliary storage* ausgelagert (*paging out*) und mit den Daten des wichtigeren Programmes gefüllt. Die Prozesse paging in und paging out werden mit einem *least used* Algorithmus entschieden. z/OS ist bemüht immer eine gewisse Anzahl an Frames frei zu halten, da dies offensichtlich nicht immer möglich ist, gibt es das sogenannte *page stealing*. Beim page stealing nimmt z/OS Frames von aktiven Benutzern und gibt sie für andere Arbeit frei. Gute Kandidaten für das page stealing sind Frames die über einen relativ langen Zeitraum nicht aktiv waren.

Ein weiterer Prozess, den z/OS zur Verfügung stellt, ist das sogenannte *swapping*. Dabei wird ein gesamter Adressraum zwischen *real storage* und *auxiliary storage* bewegt (*swap in/out*). Das *swapping* selbst wird vom *System Resource Manager* (**SRM**) auf Empfehlungen vom WLM hin durchgeführt. Um die Sicherheit des Systems zu erhöhen gibt es die *storage protect keys* (**key**). Diese keys werden jeweils in einem *Program Storage Word* (**PSW**) gespeichert. Jeder 4 KiB Frame wird mit einem solchen key assoziiert und jeder Job besitzt ein PSW. Wenn eine Anfrage gestellt wird um ein Frame zu bearbeiten, wird der key von der Anfrage mit dem key im *real storage* verglichen. Sind diese gleich oder der key im *real storage* 0 (Master Key), dann wird der Zugriff erlaubt. Eine Lese Anfrage auf ein Frame wird immer erlaubt, es sei denn das *fetch-protection* Bit ist gesetzt. Ist dieses Bit gesetzt wird wieder geprüft ob die keys übereinstimmen oder ob das anfragende Programm in key 0 (Master Key) ausgeführt wird. Wird eine beliebige Anfrage abgelehnt, wird eine *program exception interruption* ausgelöst. [7][4]

3 z/OS Interfaces

Um Anwendungen effizient und sicher starten und verwalten zu können, stehen unter z/OS verschiedene Möglichkeiten zur Verfügung. TSO (Time Sharing Option) und TSO/E (Time Sharing Option / Extension) dient Systemadministratoren in und Systemprogammierern unter z/OS als hilfreiches Tool zum Schreiben, Ausführen und Debuggen von Batch-Jobs. TSO bietet dem Nutzer eine Kommandozeilenumgebung zur Interaktion mit dem System an, es wird jedoch meist in Verbindung mit ISPF (Interactive System Productivity Facility) verwendet. ISPF dient als Menü-gesteuertes Interface zum Erstellen und Verwalten von Dateien und Programm-Modulen, dem Betrachten und Bearbeiten von Daten sowie dem Suchen und Vergleichen. ISPF interagiert mit dem System durch das Absetzen von Kommandos welche durch die TSO-Software bereitgestellt werden. Gleichzeitig bietet z/OS die Möglichkeit Unix-ähnliche Kommandozeilen-Interfaces zum Verwalten des Systems einzusetzen. Dies ist hauptsächlich dazu gedacht, den Einstieg in die z/OS-Welt zu erleichtern. Zu den Systemen welche Umsteigern und Interessierten zur verfügung stehen gehören ISHELL sowie OMVS.[7]

4 WebSphere

IBM WebSphere ist eine Anwendungsgruppe welche eine vielzahl unterschiedlichster Produkte beinhaltet. Oft wird jedoch mit dem Begrif WebSphere auch der WebSphere Application Server bezeichnet.

4.1 WebSphere Application Server

ist ein Anwendungsserver welcher eine Laufzeitumgebung zur Ausführung von JavaEE-Anwendungen bereitgestellt. Unterstützt werden in der aktuellen Version 7.0 JavaSE (in der Version 6.0), JavaEE (in der Version 5.0), Java Servlet (in der Version 2.5), JavaServer Pages (in der Version 2.1) und Enterprise JavaBeans (in der Version 3.0).

4.2 WebSphere Portal Server

Unter einem 'Portal' oder auch 'Gate' versteht man einen einzelnen Ansatzpunkt zu einer großen Auswahl von miteinander in Verbindug stehender Informationen und Interaktionen. Der WebSphere Portal Server kann solche Portale auch für Systeme auf einem IBM Mainframe zur verfügung stellen. Er bietet eine große Anzahl an nützlichen Features sowie die unterstützung einer Versionskontrolle oder der unterstützung von Java Specification Request (JSR) und Web Services for Remote Portlets (WSRP).[8]

4.3 WebSphere Portlet Factory

Die WebSphere Portlet Factory ist eine Entwicklungsumgebung für Portale und Anwendungen innerhalb einer WebSphere Portal Umgebung.

4.4 WebSphere MQ

ist ein Message Queueing System welches die Interaktion von Anwendungsystemen über unterschiedlichste Hard- und Software hinaus gestattet. Es bietet eine Nachrichtenqueue auf die mit Hilfe einfacher Kommandos zugegriffen werden kann um neue Nachrichten einzureichen oder eine Nachricht zu entnehmen.[9]

4.5 WebSphere Studio Application Developer

ist eine auf Eclipse basierende Entwicklungs- und Debugging Umgebung für Anwendungen welche auch im WebSphere Application Server eingesetzt werden können. Mittlerweile wird das Produkt unter dem Namen Rational Application Developer weiterentwickelt.

5 DB2

ist ein relationales Datenbankmanagementsystem (RDBMS) von IBM, das für z/OS und LUW verfügbar ist.[7] Das bedeutet DB2 speichert Daten in einer relationalen Datenstruktur, Tabellen. Diese Tabellen und auch *Views* werden in einer z/OS spezifischen Struktur gespeichert, den Tablespaces. Ein solcher Tablespace kann bis zu 128 Terabyte an Daten fassen.[10] DB2 kann auf zwei verschiedene Weisen auf diesen Speicher zugreifen.

SMS Tablespaces sind leichter zu verwalten, da die Grenzen vom Betriebssystem gesetzt werden. Aber dadurch das der Speicher vom Betriebsystem verwaltet wird, liegt auch das Caching von Daten in der Hand des Betriebssystems.

DMS Tablespaces hingegen werden von DB2 verwaltet. Diese bestehen aus einzelnen Dateien, die intern von DB2, beliebig aufgeteilt werden können. Durch komplexe Cache-Strategien und Bufferpools, ist DB2, im Vergleich zum Betriebssystem, besser in der Lage zu entscheiden, welche Daten häufig genutzt werden und sich somit für das Caching eignen. Daraus folgt eine höhere I/O-Effizienz.

Wann sich welcher Tablespace am meisten eignet, ist aus Tabelle [1] zu entnehmen.

Table 1. SMS/DMS Tablespace-Vergleich

System Tablespace	SMS großer Overhead bei DMS
Benutzer Tablespace	
viele kleine Tabellen	SMS großer Overhead bei DMS
sonst	DMS höhere I/O-Effizienz
temporärer Tablespace	SMS variable Größe

Um alle Tabellen, Views, Indexes, Tablespaces, Anwendungen usw. zu finden, besitzt DB2 mehrere Tabellen die Metadaten oder Daten über alle Objekte des

Subsystems beinhalten. Der *catalog* beinhaltet Informationen über alle Objekte, wie Tabellen und Views, die mit SQL-Queries abgefragt werden können. Das *directory* hingegen enthält Informationen über alle Anwendungen und kann nicht mit SQL-Queries angesteuert werden.[7]

Beim erstellen einer neuen Benutzertabelle wird der Tabellenname, Benutzername des Erstellers, der Tablespace und die Datenbank der Tabelle, in die Tabelle SYSIBM.SYSTABLES des *catalogs* geschrieben. Alle Spalten der Tabelle werden automatisch in die Tabelle SYSIBM.SYSCOLUMNS geschrieben. Damit der Ersteller auf seine Tabelle zugreifen kann, wird zusätzlich ein Eintrag in SYSIBM.SYSTABAUTH geschrieben. Das erstellen eines Index, wird in der Tabelle SYSIBM.SYSINDEXES des *catalog* festgehalten.

Für die Minimierung teurer I/O-Aktionen, besitzt DB2 Bufferpools. Bufferpools sind Bereiche im *virtual storage*, in denen DB2 Tablespaces oder Indizes zwischenspeichert.[7] Bufferpools sind Caches zwischen DB2 und den DASDs.

Ein DBMS wie DB2 ohne Log-Dateien, die für Recovery Maßnahmen benötigt werden, ist schlicht unvorstellbar. Bei den riesigen Datenmengen, die ein Mainframe verarbeitet, werden die Log-Dateien ebenfalls riesig, weshalb es zum einen *active log* und zum anderen *archive log* gibt. Alle Ereignisse werden von DB2 in den *active log* geschrieben. Sobald dieser voll ist wird er auf Platte oder Band ausgelagert und *archive log* genannt. Damit die chronologische Reihenfolge der Logs nachvollziehbar ist, werden entsprechende Informationen in einem Bootstrap festgehalten.[7]

Für genau diese Backup und Recovery Maßnahmen besitzt DB2 verschiedene Utilities. Das COPY Utility kann von den Daten entweder vollständige oder inkrementelle Kopien erstellen. Das Utility MERGECOPY kann mehrere inkrementelle Kopien zu einer einzigen inkrementellen Kopie oder mehrere inkrementelle Kopien mit einer vollständigen Kopie vereinen.[7] Durch den regelmäßigen Einsatz von MERGECOPY kann die benötigte Arbeitszeit des Tools RECOVER verringert werden und das System schneller wieder in den effektiven Betrieb genommen werden.

Natürlich besitzt DB2 noch viele weitere Utilities die unter anderem zur Optimierung der Zugriffspfade dienen (RUNSTATS) oder einen Index auf dessen Konsistenz hin überprüfen (CHECK INDEX).[10] Durch solche Utilities wird DB2 vielseitiger, wartbarer und leichter im effektiven Einsatz gehalten.

6 Adabas

(**Ada**ptable Data **Bas**e) der Darmstädter Software AG wurde ursprünglich für z/VSE und MVS entwickelt und verfolgt seit jeher die Ziele der Hochverfügbarkeit und Skalierbarkeit, die für Mainframes von entscheidender Wichtigkeit sind. Eines der Hauptfeatures ist es, dass Adabas Daten in komprimierter Form speichert.[11] Dies hat zur Folge das bei modernen Datenbanken, die mehrere Terabyte an Daten umfassen, eine nicht unbedeutende Menge an Speicherplatz gespart wird und die Effizienz der I/O-Operationen, durch die geringere Datenmenge, gesteigert wird. Adabas ermöglicht die Nutzung von Periodengruppen

und multiplen Feldern.[12] Dies entspricht dem NF^2-Datenbankmodell (*Non First Normal Form*) welches eine Erweiterung des relationalen Datenbankmodelles ist.[13] Durch die Nutzung von Periodengruppen kann Adabas einen erheblichen Performanzvorteil gewinnen, da sämtliche Informationen in einem Datensatz enthalten sind und nicht über mehrere Tabellen verteilt sind, wie es die erste Normalform des Relationenmodells erfordern würde.

Beim Start von Adabas wird der *Adabas Nucleus* und die *I/O Buffer Area* in den Hauptspeicher geladen. Dabei erfüllt der Nucleus die Aufgaben einer Engine und die I/O Buffer Area die Aufgabe eines Cache. Die I/O Buffer Area beinhaltet dementsprechend häufig genutzte Daten, um langsame I/O-Zugriffe auf die DASDs zu minimieren. Zusätzlich werden in der I/O Buffer Area Update Datensätze gesammelt um sie später in einem Rutsch (*flush*) in die Datenbank zu schreiben. Für diese I/O-Aktivitäten können bis zu 250 Threads pro Adabas Session verwendet werden.[11]

Läuft Adabas auf einem einzelnen Betriebssystem mit einem Mehrkernprozessor, kann *Adabas Parallel Services* verwendet werden um bis zu 31 Nuclei zu einem Cluster zu verbinden. Alle Nuclei dieses Clusters greifen gleichzeitig mit Schreib- und Leserechten auf eine physikalische Datenbank zu, wobei jede Datenbank einen eigenen Cluster haben kann. Dabei kommunizieren und kooperieren die Nuclei miteinander um Prozesse wie Kompression, Dekompression, Suchanfragen, Updates, etc. parallel auszuführen. Dies erhöht natürlich den Durchsatz der Datenbank was, bei immmer größer werdenden Datenbeständen, immer mehr an Bedeutung gewinnt. Um Hochverfügbarkeitskriterien zu erfüllen, sind die Cluster so konzipiert, dass falls ein Cluster (un)vorhergesehen ausfällt, auf die Datenbank zugegriffen werden kann.[14] Sollte Adabas auf einem IBM Parallel Sysplex laufen, so ist es dank der *Adabas Cluster Services* möglich Sysplex Nuclei Cluster zu bilden. Diese Sysplex Nuclei Cluster kommunizieren dabei zwischen den einzelnen z/OS Instanzen und werden mit Hilfe des Sysplex Timers synchronisiert.[15] Mit den Adabas Cluster Services werden die Vorteile der Adabas Parallel Services mit denen eines Sysplex(s) gekoppelt.

7 IMS

(**I**nformation **M**anagement **S**ystem) besteht aus dem hierarchischem Datenbanksystem **IMS/DB**, dem Transaktionsmanager **IMS/TM** und einer Menge von System Diensten, die häufig genutzte Dienste für IMS/DB und IMS/TM bereitstellen. Das Herz eines jeden IMS Subsystems ist die *control region*, ein z/OS Adressraum. Dieser hat weitere Adressräume, die die control region mit Diensten erweitern oder in denen die IMS Programme ausgeführt werden. IMS Programme können in COBOL, OO COBOL, C, C++, Java, PL/I oder in Assembler geschrieben werden.[7]

IMS ist auf z/OS zugeschnitten. Es verwendet mehrere Adressräume in denen seine Subsysteme ausgeführt werden. Mehrere Tasks werden in jedem Adressraum ausgeführt, die, falls möglich, in mehrere Subtasks zerlegt werden. Für die Kommunikation zwischen diesen Adressräumen und zur Overhead-Minimierung,

bedient sich IMS der XM-Services und legt zusätzlich, von den Adressräumen, häufig genutzte Kontrollblöcke in die *Common System Area* (**CSA**). Darüberhinaus ist IMS z/OS Sysplex kompatibel.[7]

IMS/TM kann ohne die Datenbank betrieben werden. Benutzer im Netzwerk können Schnittstellen via WebSphere MQ, TCP/IP und Java ansprechen[16], um Zugriff auf Anwendungen zu bekommen die unter IMS laufen. Die Programme können dabei auf dem gleichen, einem anderem z/OS System oder auf nicht z/OS Plattformen liegen. IMS/TM verarbeitet Nachrichten (u.a. Transaktionen), die es über das Netzwerk empfängt, asynchron.[7]

Im IMS/DB Kontext ist eine Datenbank eine Implementierung einer Hierarchie.[7] Im Relationalen Datenbank Modell würde eine solche Datenbank einer Tabelle entsprechen. Das bedeutet, dass IMS Programme häufig auf sehr viele Datenbanken zugreifen müssen.

8 Oracle Stack

Eine Software-Stack Lösung, wie der Oracle Stack, verspricht dem Kunden den Aufwand, der für die Integration, Wartung und das Upgraden der Komponenten aufgebracht werden muss, minimal gehalten wird.[5] Dafür sorgt der Hersteller indem er Komponenten verschiedener Software-Schichten nimmt und diese aufeinander abstimmt, um eine reibungsfreiere Kommunikation untereinander zu gewährleisten.

Durch den Oracle Stack und der damit verbundenen "*all-you-can-eat*" Lizenz, [5] ist Oracle in der Lage den Kunden an sich zu binden. Dadurch erhält Oracle Supportgebühren und profitiert über einen längeren Zeitraum vom Wachstumspotenzial des Kunden, da dieser nicht so einfach einzelne Komponenten, mit Komponenten anderer Hersteller, austauschen oder vollständig vom Stack abweichen kann ohne dabei die bereits erworbenen Vorteile des Stacks zunichte zu machen. Dabei ist Oracle nicht das einzige Unternehmen das die Vorteile dieser Art von Kundenbindung erkannt hat und eine monolithische IT-Lösung anbietet.[5][6][1]

Table 2. Software Stacks[5][1]

Layer	**Oracle**	**IBM**	**SAP**
Applications	Fusion Apps	3rd Party Software	MySAP Suite
Middleware	Fusion Middleware	WebSphere	NetWeaver
Database	Oracle Database	DB2	
Management	Enterprise Manager	Tivoli	
Operating System	Oracle Linux	UNIX, z/OS, other	

Glossar

auxiliary storage
Speicher der aus DASDs besteht

APE
Authorized Program Facility

central storage
siehe *real storage*

CSA
Common System Area

DASD
Direct Access Storage Device, zusammenfassende Bezeichnung für relativ langsame sekundärspeicher bzw. Externspeicher wie z.B. Magnetplatten

DAT
Dynamic Address Translation

DMS
Database Managed Storage

IMS
Information Management System

IMS/DB
Information Management System Database Manager

IMS/TM
Information Management Transaction Manager

ISPF
Interactive System Productivity Facility

LUW
Linux Unix Windows

NF^2
Non First Normal Form

PSW
Program Storage Word

real storage
Ist der Hauptspeicher/Arbeitsspeicher auf den Books

RTM
Recovery Termination Manager

SMS
System Managed Storage

SRB
Service Request Block

SRM
System Resource Manager

The Line
Ist eine Bezeichnung für die obere Grenze des Speichers die mit 16 bit adressiert werden ann (2^{16} Adressen)

TSO/E
Time Sharing Option / Extension

WLM
Workload Manager
XM
Cross-Memory

References

1. J. Gould. Evolution of the Indian IT Market: The Virtues of Intelligent Design. http://www.icrier.org/pdf/12march10JeffGould.pdf, März 2010.
2. W. Greis. Mainframe Intro/Historie. http://wwwlgis.informatik.uni-kl.de/cms/fileadmin/users/kschmidt/mainframe/wolframgreis/Mainframe_Intro.pdf, September 2010.
3. W. Greis. Mainframe Mythen. http://wwwlgis.informatik.uni-kl.de/cms/fileadmin/users/kschmidt/mainframe/wolframgreis/MainframeMythen.pdf, September 2010.
4. L. Kühner. z/OS Overview. http://wwwlgis.informatik.uni-kl.de/cms/fileadmin/users/kschmidt/mainframe/lutzkuehner/Chapter03_zOS_Intro_Foils.pdf, September 2010.
5. M. LaMonica. Software's 'stack wars'. http://news.cnet.com/Softwares-stack-wars/2100-1012_3-6062557.html, April 2006.
6. M. LaMonica. Software's 'stack wars'. http://news.cnet.com/Softwares-stack-wars—page-2/2100-1012_3-6062557-2.html, April 2006.
7. W. O. Mike Ebbers, John Kettner. *IBM Redbook Introduction to the New Mainframe: z/OS Basics*. IBM, August 2009.
8. C. D. Nagraj Alur, Isaac Allotey-Pappoe. *WebSphere Portal and DB2 Information Integrator: A Synergetic Solution.* IBM, März 2004.
9. http://en.wikipedia.org/wiki/IBM_WebSphere, 2010.
10. http://de.wikipedia.org/wiki/DB2, 2010.
11. http://documentation.softwareag.com/adabas/ada822mfr/adamf/concepts/cfadais.htm, 2010.
12. http://de.wikipedia.org/wiki/Adabas, 2010.
13. http://de.wikipedia.org/wiki/Relationale_Algebra, 2010.
14. http://documentation.softwareag.com/adabas/ada822mfr/adamf/concepts/cfoptext.htm#asm, 2010.
15. http://documentation.softwareag.com/adabas/ada822mfr/adamf/concepts/cfoptext.htm#als, 2010.
16. http://de.wikipedia.org/wiki/Information_Management_System, 2010.

IBM-Zertifizierungen

Seminar zur Kaiserslautern Mainframe Summit

Daniel John, Lars Scherer
d_john@cs.uni-kl.de, lscherer@cs.uni-kl.de.

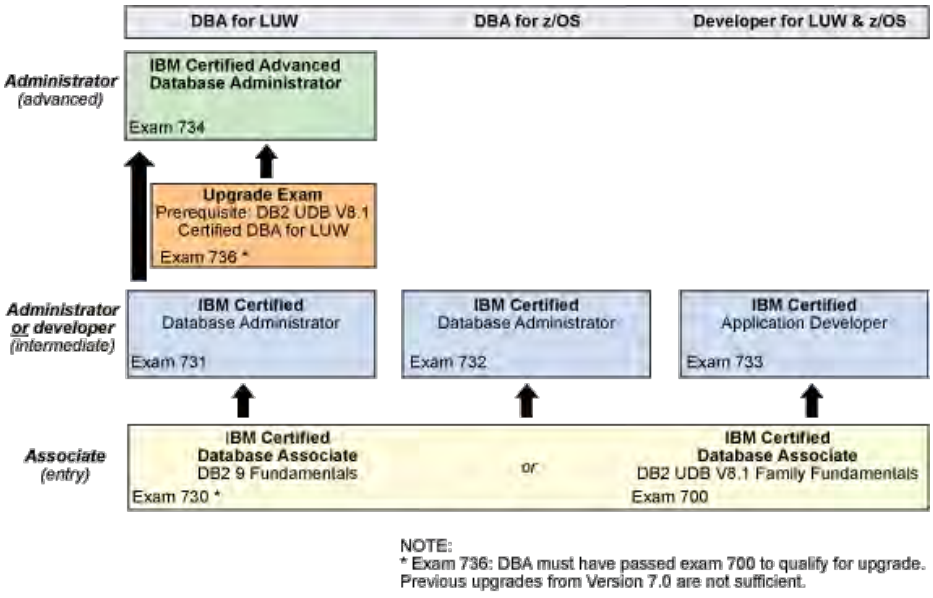
Fachbereich Informatik
Technische Universität Kaiserslautern
Betreuer: Karsten Schmidt

Abstract. Dieses Paper stellt Informationen zu IBM-Zertifizierungen für die Teilnehmer des Kaiserslautern Mainframe Summit 2010 zusammen.

Keywords: IBM, DB2, z/OS, Kaiserslautern Mainframe Summit 2010, Zertifizierungen.

1 Welche Zertifizierungen werden von IBM angeboten?

IBM bietet verschiedene Zertifizierungen, die teilweise aufeinander aufbauen (siehe Grafik, [2]).



Als Einstieg bietet IBM die Zertifizierungen 730 und 700, wobei die 700er Ende dieses Jahres ausläuft und durch die 730er ersetzt wird. Diese dienen als Grundlage und Voraussetzung für die Zertifizierungen 731,732 und 733, welche auf die Besonderheiten ihrer jeweiligen Plattform abzielen (Linux/Unix/Windows (LUW) auf der einen Seite und z/OS auf der anderen Seite). Für die LUW-Welt gibt es darauf aufbauend weitere Zertifizierungen.

2 Wann und wo finden die Zertifizierungen statt?

Ein genauer Termin steht noch nicht fest. Die Zertifizierungen sollen jedoch Ende November in Kaiserslautern stattfinden (genaueres auf der Homepage des KMS[1] sobald bekannt).

3 Was kosten die Zertifizierungen?

Die 730-Zertifizierung ist kostenlos. Alle anderen kosten voraussichtlich 30 $. Diese können sicher mit Visa-Kreditkarte, evtl. auch mit anderen Kreditkarten gezahlt werden. Eine Barzahlung oder mit EC-Karte ist nicht möglich.

4 Welche Zertifizierungen kommen für uns in Frage?

Es gibt die Möglichkeit die Zertifizierungen 730, 732 und evtl. 731 abzulegen. Diejenigen, die bereits im Wintersemester 08/09 die Zertifizierungen 730 abgelegt haben, können direkt die Zertifizierungen 731 und/oder 732 angehen. Alle anderen müssen zunächst die Zertifizierung 730 ablegen und können bei Bestehen direkt im Anschluss aufbauende Zertifizierungen erwerben.

5 Auf welche Schwierigkeitsniveaus muss ich mich einstellen?

Die Zertifizierung 730 (IBM Certified Database Associate - DB2 9 Family Fundamentals) behandelt DB2-Grundlagen (wie der Name Fundamentals schon sagt) und wurde bei der letzten Zertifizierung 2009 von allen Kaiserslauterer Teilnehmern bestanden (wenn auch teilweise erst im zweiten - direkt anschließenden - Versuch). Mit Vorlesungswissen aus Informationsysteme und/oder Datenbankanwendung sowie dem Durcharbeiten des IBM-Tutorials [730-3], sollte ein Bestehen für die meisten ohne größere Probleme möglich sein.

Bei erfolgreichem Abschneiden kann direkt im Anschluss die Zertifizierung 732 (IBM Certified Database Administrator) abgelegt werden, worauf die zweite Woche des Kaiserslautern Mainframe Summit als Vorbereitung diente. Aufgrund des engen Zeitrahmens konnten hier jedoch nicht alle Themen vollständig abgedeckt werden.

Wer das empfohlene 700-Seiten-Buch sorgfältig durcharbeitet, sollte auch hier auf der sicheren Seite sein. Mit anderen Worten: die 732-Zertifizierung ist deutlich anspruchsvoller und sollte gewissenhaft vorbereitet werden.

Neben dem Verstehen des Stoffes, ist auch stupides Auswendiglernen von etwa ZPARMs und Benutzerrechte (SYSADM, SYSOPR, ...) notwendig. Ob die Zertifizierung 731 (IBM Certified Database Administrator for Linux UNIX and Windows) abgelegt werden kann, wird momentan noch geklärt.

6 Wie sind die Zertifizierungen aufgebaut?

Zertifizierungen bestehen aus einem Multiple-Choice-Test mit 60-70 Fragen, von denen knapp 60% richtig beantwortet werden müssen, um ein erfolgreiches Abschneiden zu gewährleisten. Im Allgemeinen gibt es nur eine Antwortmöglichkeit. Bei einigen wenigen Fragen ist ein Ankreuzen mehrerer Antworten zum korrekten Beanworten der Frage notwendig.

Die Tests sind in einzelne Kapitel unterteilt, die Anhaltspunkte über Inhalt und Gewichtung der relevanten Themen liefern können (siehe Abschnitt über Test 730 oder [730-2]).

7 Tipps zur Vorbereitung auf die Zertifizierungen

Zur Vorbereitung auf die Zertifizierungen bieten sich verschiedene Ressourcen an:

Neben (teuren) Büchern gibt es von IBM online verschiedene Tutorials, Information Center (etwa zu DB2 auf z/OS [3]) und (dickere) Redbooks. Letztere sind Dokumentationen zu spezifischen Themen, die auf Erfahrungen aus der Praxis aufbauen.

8 Materialien für den Test 730 (DB2 9 Family Fundamentals)

Im Folgenden Informationen zum DB2 9 Family Fundamentals sowie Bücherempfehlungen und Links.

8.1 Test-Informationen:

- 64 Fragen
- Zeit: 90 Minuten
- Bestehensgrenze: 59%

This Database Associate is an entry level DBA or a user of any of the DB2 family of products. This individual is knowledgeable about the fundamental concepts of DB2 9 through either hands on experience or formal and informal education. The database

associate should have an in-depth knowledge of the basic to intermediate tasks required in day-to-day administration, basic SQL (Structured Query Language), understand how DB2 9 is packaged and installed, understand how to create databases and database objects, and have a basic knowledge of database security and transaction isolation[730-2].

Section 1 - Planning (14%)
Section 2 - Security (11%)
Section 3 - Working with Databases and Database Objects (17%)
Section 4 - Working with DB2 Data using SQL (23.5%)
Section 5 - Working with DB2 Tables, Views and Indexes (23.5%)
Section 6 - Data Concurrency (11%)
[730-2]

8.2 Bücher:

- DB2 9 Exam 730 Practice Questions, 18.90 €,ISBN 978-0557033188
- DB2 9 Fundamentals Certification Study Guide, 44.50 €, ISBN 978-1583470725
  (Preise von amazon.de)

8.3 Links zum Exam 730:

730-1	IBM Übersicht Exam 730 http://www-03.ibm.com/certify/tests/ovr730.shtml
730-2	IBM Inhalt Exam 730 (Kapitelübersicht) http://www-03.ibm.com/certify/tests/obj730.shtml
730-3	IBM Tutorial Exam 730 http://www.ibm.com/developerworks/offers/lp/db2cert/db2-cert730.html?S_TACT=105AGX19&S_CMP=db2certlp
730-4	Testfragen zum 730er http://www.test-inside.com/000-730.htm
730-5	Noch mehr Testfragen zum 730er http://pdf.test-inside.com/000-730.pdf
730-6	PDF's zur Vorbereitung vom WS2008/09 http://wwwlgis.informatik.uni-kl.de/cms/courses/db2zertifizierung/

9 Materialien für den Test 732 (DB2 9 Database Administrator for z/OS)

Im Folgenden Informationen zum DB2 9 Database Administrator for z/OS sowie Bücherempfehlungen und Links.

9.1 Test Informationen:

- 69 Fragen
- Zeit: 90 Minuten
- Bestehensgrenze: 57%

The test contains five sections totalling approximately 69 multiple-choice questions. The percentages after each section title reflect the approximate distribution of the total question set across the sections.[732-2]

Section 1 - Database Design and Implementation (30.5%)
Section 2 - Operation and Recovery (29%)
Section 3 - Security and Auditing (7%)
Section 4 - Performance (29%)
Section 5 - Installation and Migration / Upgrade (4.5%)
[732-2]

9.2 Bücher:

- DB2 9 for Z/OS Database Administration: Certification Study Guide, 50.99 €, ISBN 978-1583470749
  (Preise von amazon.de)

9.3 Links zum Exam 732:

732-1	IBM Übersicht Exam 732 http://www-03.ibm.com/certify/tests/ovr732.shtml
732-2	IBM Inhalt Exam 732 (Kapitelübersicht) http://www-03.ibm.com/certify/tests/obj732.shtml
732-3	Testfragen http://pdf.test-inside.com/000-732.pdf
732-4	Noch mehr Testfragen http://www.itexamstube.com/exam.asp?e=1189

732-5	PDF's von der IBM-Webseite: http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc/db29lib.htm
732-6	DB2 Version 9.1 for z/OS Administration Guide http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.admin/dsnagk16.pdf?noframes=true
732-7	DB2 Version 9.1 for z/OS Performance Monitoring and Tuning Guide http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.perf/dsnpfk17.pdf?noframes=true
732-8	Redbook: DB2 UDB for z/OS Version 8 http://www.redbooks.ibm.com/redbooks/pdfs/sg246079.pdf
732-9	Redbook: DB2 for z/OS Stored Procedures http://www.redbooks.ibm.com/redbooks/pdfs/sg247083.pdf

10 IBM System z Mastery Test

Allgemeine Informationen zum z Mastery Test finden sich unter [z-1]. Der Test prüft Basiswissen zu z/OS [z-2]. Trainingsmaterial findet sich unter [z-3].
Der Test kostet normalerweise 95 $. Man kann sich bei IBM unter [z-4] registrieren und erhält einen Voucher, der eine kostenlose Teilnahme am Test bis Jahresende ermöglicht.
Der Test kann bei einer mit IBM zusammenarbeitenden Zertifizierungsfirma (Testcenter) abgelegt werden. Von den mehr als 200 Testcentern in Deutschland sind die in Baumholder, Mannheim, Mainz, Darmstadt, Karlsruhe und Darmstadt am nächsten. Diese findet man über Prometric [z-5], einem weltweiten Anbieter von Test und Assessment Services. Zunächst muss man sich jedoch auch bei Prometric registrieren (am besten online oder telefonisch 0800 1839708) und danach einen individuellen Termin mit einem der Testcenter vereinbaren.

Links zu z Mastery Test:

z-1	Allgemeine Informationen zum z Mastery Test http://www.ibm.com/developerworks/university/systemz/masterytest/index.html
z-2	Inhalte des z Mastery Test http://www-03.ibm.com/certify/mastery_tests/objZ01.shtml
z-3	Redbook zum z Mastery Test http://www.redbooks.ibm.com/abstracts/sg246366.html
z-4	Registrierung bei IBM http://www.ibm.com/developerworks/university/systemz/masterytest/students.html#register
z-5	Prometric http://www.prometric.com/default.htm

11 Links

1	Homepage Kaiserslautern Mainframe Summit http://wwwlgis.informatik.uni-kl.de/cms/courses/mainframesummit/
2	IBM Zertifizierungen http://www.ibm.com/developerworks/data/library/techarticle/dm-0401fosdick/
3	Information Center for DB2 z/OS: http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp
4	Everything you need to know about DB2 Certification http://download.boulder.ibm.com/ibmdl/pub/software/dw/data/dm-0401fosdick/dm-0401fosdick-pdf.pdf
5	Zertifizierungsinformationen der Uni Halle 2008 http://dbs.uni-leipzig.de/file/db2_zert.pdf
6	PDF's von der IBM-Webseite: http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc/db29lib.htm

12 Webtool zum Trainieren mit Testfragen

Als Ergänzung zu diesem Dokument wurde eine Website implementiert, die als Übungsseite genutzt werden kann. Dort ist es möglich zu den verschiedenen Zertifizierungen Fragen und Antworten einzutragen und auch zu bearbeiten.
In einem Testdurchlauf kann man die Multiple-Choice-Fragen dann beantworten und bekommt beim Abschluß eine Statistik mit der Anzahl an Fehlern, den richtigen Antworten und der benötigten Zeit angezeigt.
Die Seiten wurden in PHP programmiert und als Datenbasis dient eine MySQL-Datenbank.
Die Seiten sind nur aus dem IP-Bereich der Universität erreichbar und zusätzlich durch einen Passwortschutz gesichert. Die Zugangsdaten können beim Seminarbetreuer oder den Autoren erfragt werden.

Data Sharing in einer z/OS-DB2 Umgebung

Roya Parvizi and Sebastian Ebert

Lehrgebiet Informationssysteme
Technische Universität Kaiserslautern
Betreuer: Dipl.-Inf. Karsten Schmidt

Abstrakt Hochverfügbarkeit von Rechenzentren ist heute ein größeres Thema, als je zuvor. Dieser Artikel behandelt einen IBM-Ansatz dieses Problem zu adressieren, das sogenannte Data Sharing (DS). Hierbei handelt es sich um, durch die Hardware und das Betriebssystem z/OS unterstützte, Mechanismen zur Behandlung von Ausfällen und Nebenläufigkeiten. Am Beispiel des Datenbankmanagementsystems DB2 wird erläutert, welchen Weg IBM bei der Umsetzung geht.

1 Einleitung

In Zeiten, in denen es eine Bank Millionen Euro kostet, wenn ihre Systeme nur für wenige Stunden ausfallen, ist es notwendig, Maßnahmen zu ergreifen, die solchen Situationen entgegenwirken. IBM erreicht mit seiner System z Architektur in der größten Ausbaustufe eine Ausfallsicherheit von 99,999%, was etwa 5 Minuten pro Jahr entspricht. Dies wird erreicht, indem jedes Bauteil innerhalb eines Mainframes mehrfach vorhanden ist. Zudem wird der komplette Mainframe selbst, ebenfalls in mehreren Kilometern Entfernung ein zweites mal vorgehalten.

Im Hardwarebereich wird somit alles mögliche getan, um einen Ausfall des Systems zu verhindern. Doch wie sieht es dabei mit den Anwendungen auf dem Mainframe aus? Eine der Hauptanwendungen ist DB2, eine highend Datenbanklösung von IBM. Diese Arbeit beschäftigt sich mit dem sogenannten DS von DB2. Neben der angesprochenen, erhöhten Ausfallsicherheit bietet das Verfahren weitere Vorteile, wie z.B. verbessertes Workload Balancing und eine verbesserte Erweiterbarkeit.

Dieser Artikel gibt einen Überblick über Data Sharing in DB2 auf System z. Dabei werden die grundlegenden Begriffe definiert, die benötigte Architektur beschrieben und weitere Details bezüglich des Datenbankmanagementsystems gegeben. Anders als [BAP+06] handelt es sich hierbei nicht um ein bis ins Detail beschriebenes Handbuch zur Einrichtung von Data Sharing. Stattdessen beabsichtigt die Arbeit einen Überblick über das Thema zu geben, um auf dessen Basis weitere Nachforschungen durchführen zu können.

Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

Kapitel 2 beschreibt Grundlagen des Data Sharing, sowie die dafür benötigte Hardware-Architektur.

Kapitel 3 behandelt die Thematik des Data Sharing an Hand einer Anwendung, in diesem Fall des Datenbankmanagementsystems IBM Database2.

2 Grundlagen und Architektur

Das folgende Kapitel definiert den Begriff Data Sharing und gibt einen Überblick über die benötigte Architektur des Gesamtsystems.

2.1 Data Sharing

Data Sharing (DS) bedeutet das parallele Zugreifen (lesen und schreiben) von mehreren Applikationen auf einen (mitunter verteilt vorliegenden) Datenbestand. Um einen Überblick über paralleles Arbeiten zu bekommen, zeigt Abbildung 1 die verschiedenen Ausführungen paralleler Datenbank Architekturen (Abbildungen und Informationen siehe [BAP+06]).

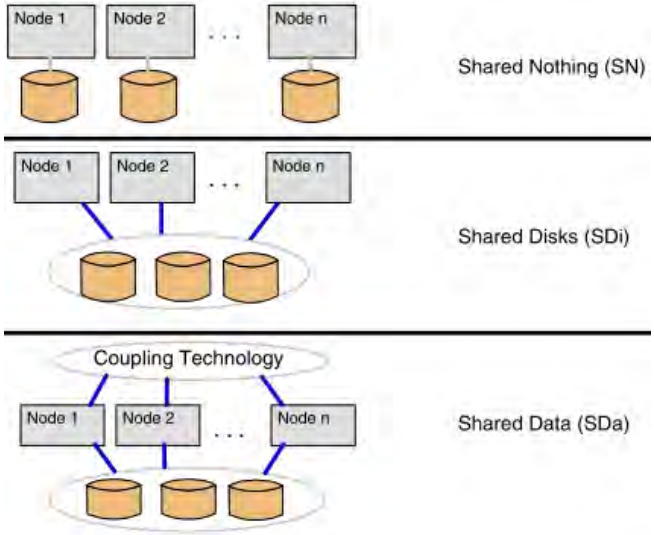


Abbildung 1: Parallele Datenbank Architekturen: *v.o.n.u.: a) keine Daten-Teilung, b) geteilte Datenspeicher, c) geteilte Daten*

Shared Nothing (SN) Vorteil dieser Methode ist, dass Lasten verteilt werden können. Dazu werden die Daten partitioniert, was zum Beispiel nach der Performance von Knoten geschehen kann. Jeder Netzwerkknoten verwaltet seine eigenen Daten. Dabei hat er kein Wissen über weitere Knoten und

somit keine Informationen über Daten, die auf diesen Knoten vorhanden sind. Aus diesem Grund muss sich die Anwendung um die Verknüpfung der Daten kümmern. Das bedeutet, dass jede Anwendung, die auf den Daten arbeiten will, die Partitionierung kennen muss. Soll nun ein weiterer Knoten hinzugefügt werden, müssen die Daten neu partitioniert und somit alle Anwendungen angepasst werden. Zweiter großer Nachteil von SN ist die Notwendigkeit eines verteilten commits, da geänderte Daten nicht notwendigerweise nur auf einem Netzwerkknoten liegen müssen. Als letzter Punkt muss auf die mangelnde Ausfallsicherheit hingewießen werden. Sobald ein Knoten ausfällt, sind die Daten darauf für die Anwendung nicht mehr zu erreichen. Ein weiteres Arbeiten ist somit unter Umständen nicht mehr möglich.

Shared Disks (SDi) Während im SN Modus keinerlei Datenteilung innerhalb der Knoten vorhanden ist und jeder Knoten nur auf seinem eigenen Datenbestand arbeiten kann, ist das im SDi Modus anders. Jeder Knoten hat Zugriff auf alle Daten, da die Datenspeicher geteilt werden. Fällt ein Knoten aus, so ist ein weiteres Arbeiten dennoch möglich. Vorteil dieser Methode ist das verbesserte Workload Balancing, das Verteilen von Aufgaben, je nach Auslastung der einzelnen Knoten. Von Nachteil dabei ist, dass Lock- und Buffer-Informationen durch die Knoten selber ausgetauscht werden müssen. Das bringt Performance-Probleme mit sich, je mehr Knoten an dem Verbund beteiligt sind.

Shared Data (SDa) SDa bietet genau wie SDi ein gutes Workload Balancing, da mehrere Netzwerkknoten auf die gleichen Daten zugreifen und somit die Last anhand der Auslastung der jeweiligen Knoten verteilt werden kann. Im Gegensatz zu SDi kommt allerdings eine weitere Komponente, die sogenannte *Coupling Facility (CF)* (die CF unterliegt seit 1994 einem IBM Patent, vgl. [EJM+94]) ins Spiel. Diese sorgt dafür, dass sich die Knoten nicht mehr untereinander über Locking und Buffering austauschen müssen, sondern sich stattdessen an sie wenden. In der CF werden alle Lock- und Buffer-Informationen, die in der DS zur Verfügung gestellt wurden, gesammelt abgelegt, sodass jeder Knoten nur noch mit ihr kommunizieren muss. Das kann für einen erheblichen Performance-Gewinn sorgen.

2.2 Vorteile Data Sharing

Verbesserte Datenverfügbarkeit Durch die Tatsache, dass mehrere Pfade von der Applikation zu den Daten vorhanden sind, ergibt sich eine höhere Verfügbarkeit des Systems. Das begründet sich dadurch, dass der Ausfall eines Knotens nicht zum Stillstand des gesamten Systems führt. Es ist immer noch ein weiterer Pfad zu den Daten verfügbar.

Verbessertes Workload Balancing, verbesserte Erweiterbarkeit Ohne DS ist Workload Balancing nur möglich, indem ein weiterer Knoten zum System hinzugefügt wird, der eine Kopie der vorhandenen Daten erhält, da diese wie oben beschrieben, nicht geteilt werden. In einem DS-Verbund kann jedoch ohne große Probleme ein weiteres Mitglied in das System eingefügt

werden und kann fortan genutzt werden, um die Kapazitäten des Gesamtverbunds besser zu verteilen. Kommt es zu einem Engpass im System, kann ohne Weiteres ein neuer Knoten hinzugefügt werden, um so das Gesamtsystem zu entlasten. Somit bietet DS eine verbesserte Erweiterbarkeit.

Höhere Transaktionsraten Durch das verbesserte Workload Balancing ist es möglich, die Transaktionsraten für das Gesamtsystem zu erhöhen. Das geschieht automatisch dadurch, dass nicht ein Netzwerkknoten einen Großteil der Aufgaben erfüllen muss, weil z.B. die Daten bei ihm liegen (Shared Nothing), sondern die Aufgaben gleichmäßig auf alle Teilnehmer verteilt werden können.

Transparenz Während beim Shared Nothing Ansatz die Applikationen an eine neue Systemkonfiguration angepasst werden müssen, ist das mit DS nicht nötig. Jegliche Konfiguration des DS-Verbunds geschieht transparent für die Anwendung, ein Anpassen dieser ist nicht weiter erforderlich.

2.3 Sysplex und Parallel Sysplex

Nachdem nun ein einführender Überblick über das Thema Data Sharing gegeben wurde, wird im Folgenden die Architektur näher beleuchtet. Bruni et al. beschreiben einen *Sysplex* als eine Gruppe von z/OS Systemen, die mittels spezialisierter Hard- und Software miteinander kommunizieren und kooperieren [BAP+06]. Dabei kommt ein Sysplex Timer zum Einsatz, der die Mitglieder des Verbunds synchronisiert. Die Kommunikation der einzelnen Mitglieder untereinander erfolgt via *Enterprise Systems Connection* (ESCON) oder *Fibre Connection* (FICON).

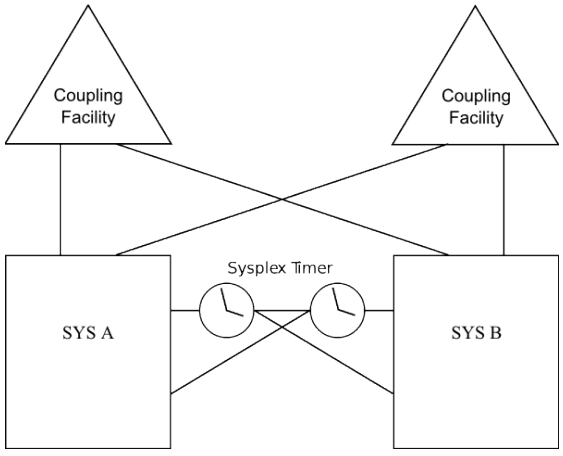


Abbildung 2: Parallel Sysplex: *Parallel Sysplex mit Komponenten Sysplex Timer und Coupling Facilities*

Kommt zu der Konfiguration noch mindestens eine CF hinzu, spricht man von einem Parallel Sysplex (für einen Überblick über die Architektur, siehe

Abbildung 2, angelehnt an [Vet08]). Die Vorteile sind verbessertes Caching und Locking für alle Mitglieder des Verbunds.

Sysplex Timer Der Sysplex Timer synchronisiert die Zeit innerhalb des Sysplexes. Das ist notwendig, da z.B. für das Logging genaue Zeiten absolut unerlässlich sind. Dabei kann es sich sowohl um dedizierte Hardware handeln, als auch um eine in ein System z eingebaute Komponente [BZK10]. Dieser sollte zum Zwecke erhöhter Ausfallsicherheit mehrfach vorhanden sein.

Coupling Facility Die Aufgabe der *CF* ist es, in einem Parallel Sysplex Locking und Buffering zentral zu verwalten. Jedes Mitglied des Verbunds muss sich somit nur noch mit dieser Komponente synchronisieren, was einen Performance-Gewinn gegenüber dem SDi Ansatz bringt, bei dem sich alle Komponenten untereinander austauschen müssen.

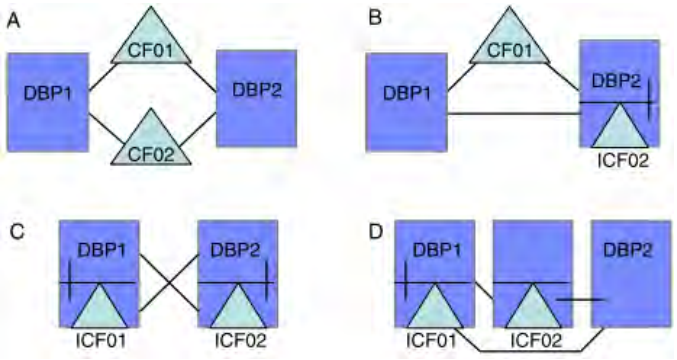


Abbildung 3: Mögliche CF Konfigurationen: *A) 2 externe CFs, B) 1 interne + 1 externe CF, C) 2 interne CFs, D) 2 interne CFs, mit 1 außerhalb der Verbund-Mitglieder*

Auf der CF läuft ein eigenes Betriebssystem, dass sich lediglich um die angesprochenen Aufgaben kümmert. Dabei kann es sich um externe dedizierte Hardware oder interne Komponenten handeln. Einen Überblick über die vier verschiedenen Konfigurationsmöglichkeiten bietet Abbildung 3. Konfiguration A zeigt zwei externe CFs. Extern bedeutet dabei, dass sie extern der DB2 Mitglieder liegen. B stellt eine externe, sowie eine in einem Mitglied des Verbunds vorhandene CF dar. Die am Häufigsten auftretende Variante is C [BAP+06], hier sind beide CFs innerhalb der Mitglieder vorhanden. Die letzte Konfiguration D zeigt erneut zwei interne CFs, wobei eine der beiden nicht auf einem Verbunds-Mitglied läuft.

Es gibt zwei Vorteile, die CF in einem Mitglied laufen zu lassen. Erstens kann dadurch ein ICF (Internal Coupling Facility) Prozessor verwendet werden, der finanzielle Vorteile gegenüber einem Standard-Prozessor bietet. Zudem sind die Kommunikationswege kürzer, was einen Geschwindigkeitsvorteil mit sich bringt.

Egal welche der vorgestellten Konfigurationen gewählt wird, die CF sollte immer dupliziert werden, damit im Falle eines Ausfalls der Sysplex weiter arbeiten kann.

3 Data Sharing mit DB2

3.1 Implementierung

Die Implementierung des Data Sharing (DS) umfasst drei Komponenten [Kum96]:

- DB2-Mitglieder benutzen die **Shared Communication Area (SCA)**, (auch bekannt als *List Structure*), um die Kontrollinformationen an die anderen Mitglieder der DS Gruppe zu übertragen. Die SCA enthält alle notwendigen Informationen für die Wiederherstellung der DS-Gruppe.
- Die **Lock Structure** besteht aus zwei Teilen: Eine Coupling Facility Lock Table, bekannt als *Modified Resource List*, und eine Coupling Facility Lock Hashtabelle, bekannt als *Lock Table*. Die *Modified Resource List* wird für die Speicherung von Locks benutzt und schützt die Daten, die geändert werden sollen. Es beinhaltet die Namen und Lock Status der Resourcen. Die Lock Table enthält Lock-Statusinformationen und die Mitglieder, denen die Locks gehören. Die Lock Table wird für globale Lock-Serialisierung benutzt.
- Die veränderten Seiten oder Daten müssen, bevor sie auf das Direct Access Storage Device (DASD) gespeichert werden, in die **Group Buffer Pools (GBPs)** geschrieben werden. Jedes DB2-Mitglied muss dann die Seiten entweder von dem DASD oder dem GBP nochmal lesen, wenn es erneut auf die Seite zugreifen will.

Diese drei Strukturen können doppelt in zwei verschiedene CFs gespeichert werden. Dabei würde eine als Primary und die andere als Secondary fungieren. Dies hat den Vorteil, dass wenn die Primary CF ausfällt, automatisch auf die Secondary CF umgestellt werden kann. Dafür ist kein Rebuild notwendig. Allerdings ist die Dauer für ein Rebuild von der SCA und der Lock Structure im Gegensatz zu den GBPs minimal, so dass das sogenannte *Structure Duplexing* nur für GBPs wichtig ist.

Der Aufbau und Zusammenhang der Komponenten innerhalb einer Data-Sharing Gruppe ist in Abbildung 4 (angelehnt an [BZK10]) verdeutlicht. Das Prinzip des Group Buffer Pool wird im Folgenden etwas genauer erläutert.

Group Buffer Pools Die *GBPs* oder *Cache Structure* sind Strukturen innerhalb der CF-Struktur, die das gemeinsame Benutzen von Daten zwischen mehreren Subsystemen innerhalb einer DS-Umgebung ermöglicht. Sobald ein Mitglied der Gruppe eine Seite liest, wird diese Seite in den Virtual Buffer Pool (Local Buffer Pool) des Mitglieds eingelesen. Wenn die Seite aktualisiert wird, wird die veränderte Seite durch ein *force-at-commit* in den *Group Buffer Pool* geschrieben. Zusätzlich wird sie in den Virtual Buffer Pools der anderen Mitglieder der Gruppe als ungültig markiert. Dieser Prozess wird *Cross-Invalidation* genannt und stellt sicher, dass alle Mitglieder mit der aktuellsten Version der Seite arbeiten. Der Ablauf ist in Abbildung 5 (nach [LL08]) dargestellt.

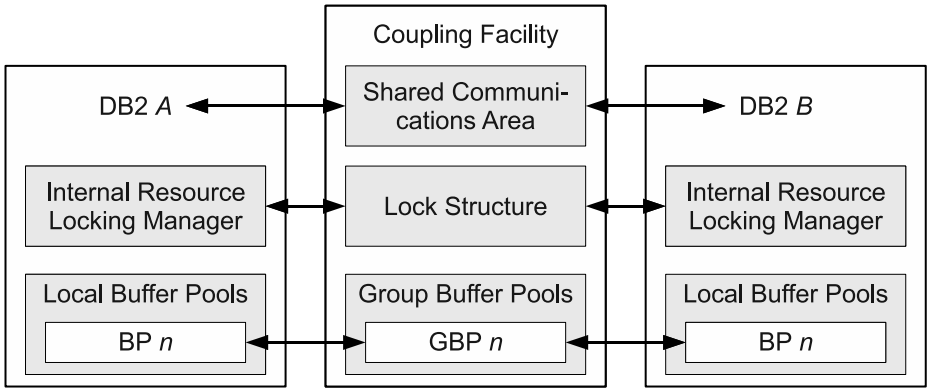


Abbildung 4: *Die Komponenten des DB2 Data-Sharing.*

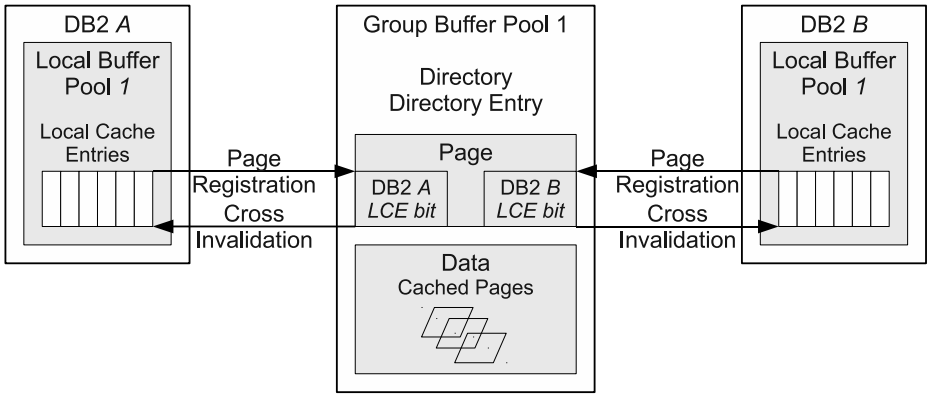


Abbildung 5: *Das Prinzip des Group Buffer Pools.*

Duplexing in Group Buffer Pools Wenn es ein Primary und Secondary gibt, werden die Pages in folgender Art und Weise geschrieben:

1. Für eine Anzahl an Pages die geschrieben werden sollen:
   (a) Jede Page wird *asynchron* in den Secondary GBP geschrieben
   (b) Jede Page wird *synchron* in den Primary GBP geschrieben
2. Nachdem alle Pages in Primary GBP geschrieben wurden, überprüft DB2, ob alle Pages bereits in Secondary GBP vollständig geschrieben sind. Falls der Schreibvorgang noch nicht beendet sein sollte, erzwingt DB2 dessen Fertigstellung.

3.2 Mechanismen für globale Konsistenz

Die Daten, die als *sharable* definiert sind, können von jedem in der DB2 Gruppe gleichzeitig benutzt werden. Gleichzeitig können mehrere Subsysteme die gleiche

Resource bearbeiten. Dies wird durch das sogenannte *inter-DB2 read/write* ermöglicht. Veränderte Daten oder Daten, für die ein r/w Interesse besteht, werden immer in den Buffer Pool der Gruppe gespeichert. Die *Coupling Facility* kümmert sich darum, dass die geänderten Seiten in den Buffer Pools der Mitglieder als ungültig markiert werden.

Es gibt zwei Typen der Kontrolle, die die Datenkonsistenz zwischen DB2-Teilnehmern in einer DS-Gruppe kontrolliert: (i) Concurrency Controls (Nebenläufigkeit) und (ii) Coherency Controls (Zusammengehörigkeit von Daten). Um eine Concurrency Control zwischen den Mitgliedern bereitzustellen, wird eine neue Art der Locking-Struktur benötigt.

Locking Das Locking in einer DS-Umgebung unterscheidet sich von Locking in Einzelsystemen. Anstatt *Implicit Hierarchical Locking*, wird *Exiplicit Hirarchical Locking (EHL)* benutzt [LL08]. Der einzige Unterschied besteht darin, dass EHL ein Token hält, welches Eltern-Kind Beziehungen beschreibt. Ein Eltern-Lock sperrt einen Table Space oder eine Partition. Ein Kind-Lock hingegen ist für eine Seite oder eine Zeile zuständig.

Folgende verschiedene Lock-Typen existieren in einer DS-Umgebung:

Local Locks Local Locks werden in Umgebungen von Einzel-Subsystemen benutzt. Sie stellen nur intra-DB2 concurrency bereit, das heißt, dass Nebenläufigkeit nur in einem DB2 Subsystem möglich ist.

Global Locks Global Locks sind die Locks, die den DB2 Subsystemen einer DS-Gruppe bekannt gegeben werden müssen. Im Gegensatz zu Local Locks bieten Global Locks Kontrolle für sowohl intra-DB2 Nebenläufigkeit als auch für inter-DB2 Nebenläufigkeit. In einer Data-Sharing Umgebung sind fast alle Locks global. Ob ein Lock global wird, ist abhängig davon, ob es physisch oder logisch angefordert wurde.

P-Locks Physical Locks oder P-Locks werden für physische Objekte in Puffern, z.B. Table Spaces, Index Spaces und Partitions benutzt. Sie sperren also Pages oder Page-Sets der DB2 Umgebung. Ein P-Lock muss jedoch nicht auf allen Partitionen eines Page Set definiert sein. Weiterhin hat ein P-Lock auf ein Page Set nicht zwangsläufig einen P-Lock auf dem entsprechenden Index zur Folge. Sie kontrollieren den gleichzeitigen Zugriff verschiedener DS-Mitglieder auf einen einzelnen Table-Space, gehören einem DB2 Mitglied und sind übertragbar. Sie werden eher für Coherency anstatt für Concurrency benutzt. Die Coupling Facility benutzt zwei Arten von P-Locks: Page P-Locks und Page Set P-Locks.

L-Locks Das Pendant zu P-Locks sind Logical Locks oder L-Locks. Sie kontrollieren die Nebenläufigkeit von SQL-Anwendungen (siehe [BZK10]) und sperren z.B. Daten eines Cursors und treten in DS als auch in Non-DS-Subsystemen auf. Der Besitzer solcher Locks ist entweder eine Transaktion oder ein Programm (*Transactional Locks*). L-Locks sind nicht übertragbar und funktionieren wie

normale Locks in Single-Subsystem-Umgebungen, um Zugriffe auf Objekte zu serialisieren. Es gibt zwei Typen von L-Locks: Eltern- und Kind-Locks.

Zusätzlich enthält das DB2 DS zwei andere Lock-Arten: den Modify Lock und den Retained Lock.

3.3 Update-Prozess

Im Folgenden wird beispielhaft ein Update-Prozess innerhalb einer DS-Umgebung erläutert [IBM09]. Wenn eine Applikation z.B. ein Update von DB2A aus ausführt, welches Daten erfordert, die weder im eigenen Buffer Pool noch im GBP vorhanden sind, werden diese durch DB2A von der Festplatte gelesen. Danach werden diese Daten geändert und in den eigenen Buffer Pool geschrieben. Gleichzeitig sorgt eine Sperre durch DB2A dafür, dass andere Mitglieder der Gruppe diese Daten nicht gleichzeitig verändern können. Nachdem DB2A den Update-Prozess beendet hat, wird diese Sperre wieder freigegeben. Die veränderten Daten verbleiben im Buffer Pool von DB2A.

Angenommen eine andere Applikation, die auf DB2B läuft, möchte ebenfalls ein Update auf den gleichen Daten machen. DB2 erkennt das inter-DB2 Interesse an diesem Page Set und DB2A schreibt die veränderte Seite in die Group Buffer Pools (sowohl primary als auch secondary). DB2B holt sich darauf die neueste Version der Seite von dem Primary Group Buffer Pool. Nachdem die Applikation von DB2B den Commit ausgeführt hat, kopiert DB2 die Seite in den GBP. Dies hat zur Folge, dass die Seite im Buffer Pool von DB2A als ungültig markiert wird.

Wenn DB2A die Daten jetzt noch einmal lesen will, bemerkt diese, dass die Datenseite in dem eigenen Buffer Pool ungültig ist. Als Folge dessen, wird die neueste Version der Seite erneut vom Primary Group Buffer Pool gelesen.

Zum Schluss müssen noch die geänderten Seiten vom Primary Group Buffer Pool auf die Platte geschrieben werden. Dieser Prozess wird *Castout* genannt. Dabei müssen die Daten über den Adressbereich von DB2A gehen bevor sie wirklich gespeichert werden können. Der Grund dafür ist, dass es keine direkte Verbindung zwischen Coupling Facility und der Festplatte gibt. Die *Castout Engine* überträgt die veränderten Seiten von dem Group Buffer Pool in den private Adressbereich von dem aus sie auf die Platte geschrieben werden. Dieser Prozess ist in Abbildung 6 [LL08] zu sehen.

Wenn ein Group Buffer Pool doppelt vorhanden ist, werden die Daten nicht vom Secondary Group Buffer Pool auf die Festplatte geschrieben. Allerdings werden Seiten vom Secondary Group Buffer Pool gelöscht, wenn diese vom Primary Group Buffer Pool auf Platte geschrieben wurden.

3.4 Datenwiederherstellung

Die Datenwiederherstellung innerhalb einer DS-Umgebung ist etwas anders als üblich. Jedes Mitglied schreibt Logdateien in eigene Recovery Logs und *Boot Strap Data Set (BSDS)*. Zusätzlich muss es jedem Mitglied möglich sein, die

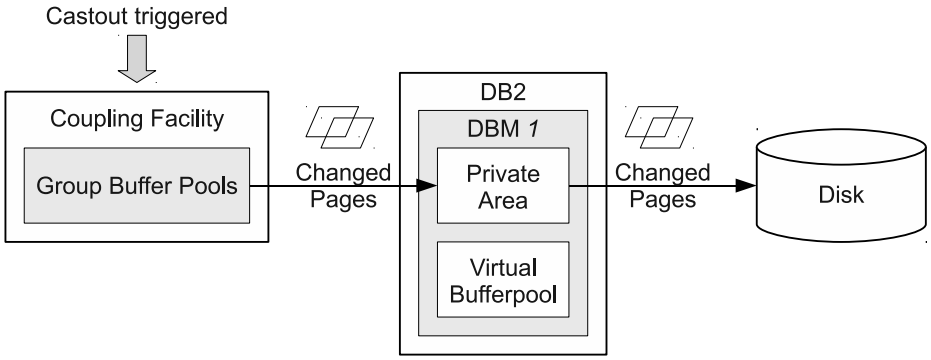


Abbildung 6: *Nachdem ein Castout ausgelöst wurde, schreibt die Castout Engine die veränderten Seiten auf die Festplatte.*

Logdateien der anderen Mitglieder zu lesen. Dies ist vor allem deswegen wichtig, da eine Wiederherstellung von Daten möglicherweise die Log Daten von mehreren DB2 Systemen benötigt. Ein Neustart der Gruppe benötigt ebenfalls Zugriff zu allen Logdateien [LL08].

Die Shared Communications Area beinhaltet die Informationen über die Logdateien und *BSDSs* aller Mitglieder der Gruppe.

Es sollte vermieden werden, die Logdateien auf ein Band zu speichern. Denn dadurch kann sich die Wiederherstellungszeit, abhängig von der Anzahl der Mitglieder der Gruppe, stark erhöhen.

3.5 Verwaltung einer Data Sharing Gruppe

Ein großer Vorteil einer DS-Umgebung ist die Möglichkeit, Workload zwischen den Mitgliedern der Gruppe zu verteilen. DB2 Subsysteme arbeiten mit der Workload Manager (WLM) Komponente von z/OS, die das optimale Balancing zwischen den Teilnehmern der Gruppe ermöglicht [LL08]. Basierend auf vordefinierten Parametern verwaltet der WLM die Arbeit entsprechend Eigenschaft und Priorität.

3.6 Sysplex Query Parallelisierung

Mit Hilfe der Sysplex Query Parallelisierung ist es möglich eine Anfrage über mehrere Mitglieder einer DS-Gruppe zu verteilen. Damit ist eine gute Skalierbarkeit für komplexe Anfragen möglich. Die Sysplex Query Parallelisierung funktioniert wie CPU Parallelität: Der *Koordinator* initiiert eine Anfrage und sendet diese zu anderen Mitgliedern der Gruppe, den *Assistenten*. Diese verarbeiten die Daten und senden das Ergebnis zurück zum Koordinator. Abbildung 7 (vgl. [LL08]) zeigt diesen Vorgang der Sysplex Query Parallelisierung.

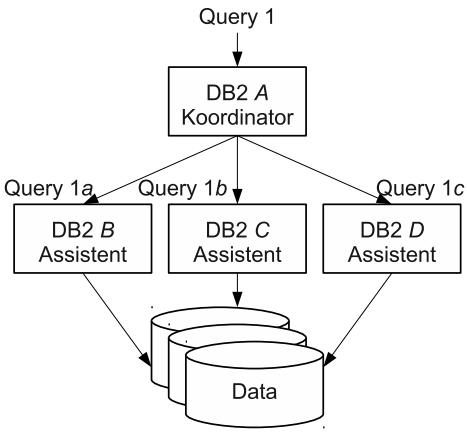


Abbildung 7: *Sysplex Query Parallelisierung*

4 Zusammenfassung

Data Sharing in DB2 erlaubt Lese- und Schreibzugriffe auf DB2-Daten von verschiedenen Subsystemen aus, die auf mehreren Central Processor Complexess (CPCs) in einem parallelen Sysplex verteilt sind.

DS bietet viele Vorteile im Einsatz von DB2 (z.B. hohe Verfügbarkeit). Allerdings ändern sich dadurch viele Verfahren, die für das Management verwendet werden, was dazu führt, dass bestehende DB2-Installationen überprüft und angepasst werden müssen. Man kann jede Applikation in eine DS-Umgebung einsetzen, wobei manche Applikationen mehr davon profitieren als andere. Um die Vorteile innerhalb einer DS-Umgebung zu erhöhen, ist es manchmal notwendig, die Applikationen anzupassen. Die Planung und die Vorbereitung für eine DS-Umgebung sind wichtige Aspekte die nicht unterschätzt werden dürfen.

Abkürzungsverzeichnis

BSDS Boot Strap Data Set. 9, 10

CF Coupling Facility. 3–6
CPC Central Processor Complexes. 11

DASD Direct Access Storage Device. 6
DS Data Sharing. 1–4, 6, 8–11

EHL Exiplicit Hirarchical Locking. 8

GBP Group Buffer Pool. 6, 7, 9

SCA Shared Communication Area. 6
SDa Shared Data. 3
SDi Shared Disks. 3, 5
SN Shared Nothing. 2, 3

WLM Workload Manager. 10

Literatur

BAP+06. Paolo Bruni, Mark Anders, KyengDong Park, Mark Rader, and Judy Ruby-Brown. *DB2 for z/OS: Data Sharing in a Nutshell*. IBM Redbooks, 2006.

BZK10. Björn Broll, Timm Zimmermann, and Tunca Karabel. Db2 for z/os. Technical report, Kaiserslautern Mainframe Summit 2010, 2010.

EJM+94. David A. Elko, John F. Isenberg Jr., Brian B. Moor, Jimmy P. Strickland, Michael D. Swanson, and George W. Wang. Method and apparatus for distributed locking of shared data, employing a central coupling facility. Technical Report 5339427, 1994.

IBM09. IBM. *Data Sharing: Planning and Administration*. IBM, 2009. SC18-9845-03.

Kum96. Ravi Kumar. *DB2 for MVS/ESA Version 4 Data Sharing Implementation*. IBM Redbooks, 1996. SG24-4791-00.

LL08. Susan Lawson and Daniel Luksetich. *DB2 9 for z/OS Database Administration: Certification Study Guide*. Mc Press, 2008.

Vet08. Andre Vetter. IBM Parallel Sysplex. Technical report, wikipedia.org, `http://en.wikipedia.org/wiki/File:GDPS.svg`, 2008. (Zugiffsdatum: 2010-10-21).

An Overview of the Support for XML in the DB2 Database System

Seminar: Mainframes Today

Caetano Sauer
csauer@cs.uni-kl.de

Databases and Information Systems Group
University of Kaiserslautern
Supervisor: Karsten Schmidt

Abstract. Due to the increasing demand for efficient and reliable management of XML data, several promising techniques arrived from the idea of integrating XML into the context of relational database systems. This paper briefly discusses early approaches to the straight-forward mapping of XML documents to relational data, further introducing the *pureXML* engine, a more sophisticated alternative which aims to integrate native support for XML into IBM's relational database system *DB2*.

1 Introduction

The widespread diffusion of XML in the domain of data management and information systems challenges both the industry, which must provide support for the standard in their products without affecting existing applications, as well as the research community, which struggles to develop efficient and reliable means of storing and managing XML data. In this context, it seems that the best solution is to simply adapt the current dominating data management technology, namely relational databases, to support storing and querying data represented as XML documents. Although, as we shall see, it is quite simple to provide a mapping from XML data and path queries to relational tables and SQL, extending the whole database architecture to actually understand and efficiently manage the hierarchical data in XML documents provides a much higher level of performance and reliability.

Section 2 presents the most fundamental differences between the XML and relational data models, justifying why it is important to have specialized (or hybrid) engines for dealing with XML. In Section 3, we present some approaches for mapping XML documents and queries into a pure relational DBMS, while Section 4 introduces a modified relational engine that can better support XML documents in its native data model. Finally, Section 5 concludes this paper.

2 XML vs. The Relational Model

The XML standard was originally developed as a format for document representation and data exchange, influenced mainly by the needs of the Web as a

platform for publishing information. Based on the fundamental syntax enforced by XML, other document formats or languages are constructed, allowing users to enhance text with structure and meaning in the form of markups.

The original goals of XML have little in common with those of a database system, but as the amounts of data stored and exchanged in the basic form of XML documents kept growing steadily over the past years—motivated primarily by the Web—, it became mandatory to efficiently manage and store XML documents in a centralized database system. It was after this point that XML documents started to be referred to as simply XML *data*, and the ideas and techniques of the relational data model—the universally accepted standard for data management—came into conflict with those of the XML data model. We present the conflicting aspects in Table 1, from the perspective which is relevant for a DBMS.

	Relational	XML
Data units	Tuples and Tables	Nodes and trees
Nesting	Flat data; Logical hierarchies with foreign keys	Natively hierarchical data with arbitrary nesting
Order	Irrelevant	Must preserve document order
Schema adherence	Mandatory	Optional
Variability	Uniform data	High degrees of variability within sub-trees
Density	Dense data, with NULL covering missing values	Sparse data, with absent or empty nodes

Table 1. Main differences between the XML and relational data models

As a consequence of the fundamental differences between data models, the need for a new query language other than SQL arrives, and this has been fulfilled by the XQuery language [4]. From a language perspective, both languages are declarative (i.e. non-procedural), which is a fundamental aspect for data independence and pipelining of large datasets through a query plan. From the typical use, on the other hand, the languages present many differences. While in SQL the triad select-project-join (SPJ) plays the main role in data operations, in XQuery the support for navigational operators is crucial.

Given all these differences, it is no wonder that developing a DBMS which can efficient, transparent, and thoroughly support XML and relational data can be very challenging. In the next chapter, we give a brief overview over some of the initial approaches to the management of XML data, which allow the straight-forward reuse of traditional RDBMS technology.

3 Managing XML in a Relational DBMS

Despite the overall diffusion of XML, the vast majority of information systems relies on relational databases, which have been intensively studied, developed, and

applied in the industry along four decades. A simple alternative to redesigning the whole database infrastructure is to provide a mapping from XML documents to the relational model, storing information about nodes and edges of an XML tree into tables of a relational schema. This technique is referred to as *shredding*. Systems that implement this approach must also provide an equivalent mapping from XML queries to proper SQL statements.

We present two basic approaches to XML shredding, the first one being what is known as the edge approach [5]. Here, the element names in a document are used to label edges of the XML tree. Using this technique, each element is represented by a label with a pair of *source* and *destination* attributes with integer values. The value for the source of a particular element is the same as the destination of its parent. Each edge is then stored in a single row of a relational table, together with a flag to indicate the kind of the destination node (element, attribute, or text), and a value column to store the text value of attribute and text nodes. In this approach, only child relations are supported for querying. For a path a/b, a self-join of the edge table is performed, selecting edges with labels a and b filtered by a predicate $a.destination = b.source$.

The second shredding alternative is the node approach, which was introduced in [13]. Its major advantage towards the edge technique is that it relies on a more powerful *node numbering scheme* [6], which allows us to also answer descendant queries. Here, the pair *(source,target)* is replaced by a triple *(start,end,level)*. The *start* and *end* attributed correspond to an ordering given to each node upon performing a depth-first search. We take a global counter and increment it for every node we visit. If going down to a child, we assign the counter value to the current *start* attribute, and if going up, assign it to *end*. Note that, for leaf nodes, *start*=*end*. Using this numbering scheme, a node b is a descendant of some node a if the interval $(b.start, b.end)$ is contained in $(a.start, a.end)$. Furthermore, if $b.level = a.level + 1$, b is also a child of a.

Further shredding techniques exist, which store one row for each path in the XML tree [11]. Other approaches rely on schema information provided by a DTD file [12], producing a more intuitive mapping between elements and tables in the relational schema. At last, we have the most simplistic approach, which is to store whole XML documents as Varchar or CLOB columns of a relational table. This approach is, however, irrelevant for our discussion, since the system is completely unaware of XML data, simply managing documents as raw character sequences. Therefore, the character data must be parsed every time a query is issued on the XML data, which is a truly deal-breaker when it comes to performance.

The biggest disadvantage of all shredding techniques is that—under the hood—the data is always stored and manipulated in pure relational form. This means that the DBMS is completely unaware of the hierarchical relationship between the shredded tuples, not allowing the use of tailored query operators such as the Structural Join [13]. Furthermore, such a shredding system is incapable of delivering more fine-grained transaction control over sub-trees in the XML document. With these concerns in mind, the native approach was introduced, storing and handling the data with complete awareness of the hierarchical struc-

ture. In the next chapter, we present a state-of-the-art implementation of the native approach: DB2's pureXML, developed by IBM.

4 pureXML: The DB2 Approach

In order to overcome all the drawbacks of shredding techniques, IBM has incorporated native XML support into its database product DB2, naming the feature as *pureXML* [3]. The storage layer of DB2's architecture was extended with a specialized XML store, which is handled separately from the standard relational store. Despite the physical separation, the two storage components are closely linked, which not only allows a smooth integration of XML and relational data in the application logic (mixed SQL/XQuery queries), but also ensures good performance levels by avoiding the mapping overhead between data models. On the following, we introduce the specialized XML storage, which is actually very similar to many other native XDBMS products such as Tamino [2], and scientific prototypes such as Natix [9].

The heart of pureXML's storage is the XML data type, which are used like any other column type such as INT or VARCHAR. This means XML documents are supported only as values in a database column, but at the storage layer they are handled separately from other column types. Each row containing an XML-typed column is then associated to a well-formed XML document (or to the NULL value to indicate the absence of a document). In order to insert a row containing an XML value, the user sends the XML data in its plain character format, which is processed by a SAX parser. Using the SAX events, a tree structure correspondent to the document is built in main memory, replacing element and attribute names by a unique integer identifier, which is mapped using a dictionary.

At the storage level, document trees are stored in a page of the specialized XML store, i.e., separate from pages containing values of relational types. If a tree is too large to fit into a page, it is split into connected sub-trees, referred to as *regions*. In order to keep track of the regions, a single *region index* is kept for each table containing one or more XML-typed columns. This means that the index keeps track of multiple regions of multiple documents. Special proxy nodes are used to identify, within the page, that the corresponding sub-tree is located in another region. Figure 1 shows an example where a single tree is split into three different regions, identified by the colors white, black, and gray. Pages can also contain regions of different documents (note that here we consider a small document as being a single region), as shown in the third page.

Inside each page, nodes are clustered in a breadth-first manner, meaning that siblings are stored next to each other. This kind of clustering allows efficient support to navigation steps like parent, child, and siblings. On the other hand, this imposes an overhead for document reconstruction, since the serialization of XML documents is performed in depth-first navigation. To efficiently answer path queries without having to traverse whole sub-trees, each page maintains

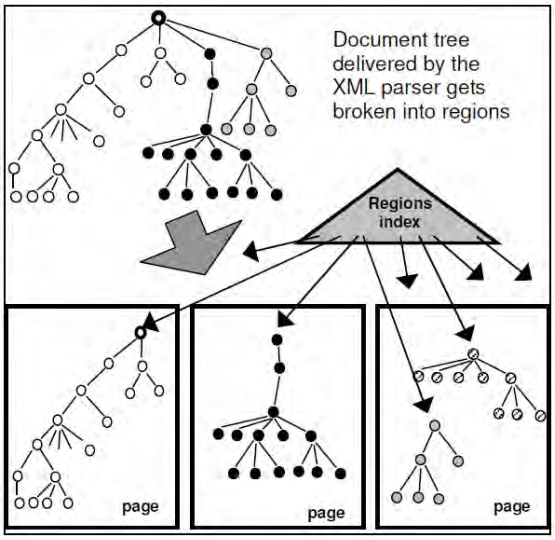


Fig. 1. Storage of a document consisting of multiple regions

a header with path information and pointers to child/parent nodes within the same page, which allows a more depth-first kind of traversal.

In order to allow the interplay of SQL and XQuery, as well as the reconstruction of XML data from relational format and vice-versa, the SQL/XML standard [7] was developed as part of SQL:2003. DB2 also provides several other features that are part of the pureXML technology, such as indexes on XML documents, support for XML Schema, a tailored XQuery optimizer, and a set of tools and utilities for applications and DB administration [3] [8] [1].

5 Conclusion

This paper analyzed the main approaches for XML data management which are integrated with relational database systems. We first provided an overview of the main differences between the XML and relational data models, which brought to attention the fact that specialized storage mechanisms and query languages must be used for each of them. In Section 3, we presented the approaches that provide a *mapping* from XML to relational data, not requiring a specialized storage component. However, because of major performance drawbacks—mostly derived from the obliviousness of the hierarchical structure of XML—, the need to actually incorporate the XML support into the storage layer of the database system was made necessary for efficient XML data management. Section 4 then introduced the pureXML approach used in the DB2 database system, focusing on the specialized storage component for XML. We explained how XML subtrees are stored in pages and which mechanisms are used to insert, navigate, and reconstruct XML documents using this native approach.

Despite all benefits and usability features provided by such an integrated Relational/XML approach of pureXML, further improvements are possible if all layers of a database system are completely redesigned for the exclusive support for XML. Storage mechanisms can manage individual nodes as the granule of data, instead of using whole regions like in pureXML. Other storage approaches make use of a path index which leads to the actual payload data, which in XML is stored at leaf nodes. Furthermore, other database components can be leveraged for XML, such as transaction support with fine-granule isolation levels for XML trees, and different kinds of XQuery compilation. We refer to [10] for related work and alternative approaches to native XML data management.

References

1. DB2 9 pureXML Guide. `http://www.redbooks.ibm.com/abstracts/sg247315.html`. (2007).
2. Tamino: The XML Database. `http://www.softwareag.com/Corporate/products/wm/tamino/default.asp`. [Online; accessed 25-Nov-2010].
3. M. N. Bert, M. Nicola, and B. Linden. Native XML Support in DB2 Universal Database. In *VLDB*, pages 1164–1174, 2005.
4. D. D. Chamberlin. XQuery: A Query Language for XML. In A. Y. Halevy, Z. G. Ives, and A. Doan, editors, *SIGMOD Conference*, page 682. ACM, 2003.
5. D. K. Daniela Florescu. Storing and querying XML data using an RDMBS. *IEEE Data Engineering Bulletin*, 22:27–34, 1999.
6. P. F. Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, STOC '82, pages 122–127, New York, NY, USA, 1982. ACM.
7. A. Eisenberg and J. Melton. Advancements in SQL/XML. *SIGMOD Rec.*, 33:79–86, September 2004.
8. K. B. et al. System RX: one part relational, one part XML. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 347–358, New York, NY, USA, 2005. ACM.
9. T. Fiebig, S. Helmer, C.-C. Kanne, G. Moerkotte, J. Neumann, R. Schiele, and T. Westmann. Anatomy of a native xml base management system. *The VLDB Journal*, 11:292–314, December 2002.
10. C. Mathis. *Storing, Indexing, and Querying XML Documents in Native Database Management Systems.* PhD thesis, University of Kaiserslautern, July 2009. Verlag Dr. Hut, München.
11. S. Pal, I. Cseri, O. Seeliger, G. Schaller, L. Giakoumakis, and V. Zolotov. Indexing xml data stored in a relational database. In *VLDB*, pages 1134–1145, 2004.
12. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. pages 302–314, 1999.
13. C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman. On supporting containment queries in relational database management systems. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, SIGMOD '01, pages 425–436, New York, NY, USA, 2001. ACM.

Schnelles Klonen

Seminar: Mainframes Today

Michael Rappold, Johannes Schildgen
{m_rappol, j_schild}@cs.uni-kl.de

Database and Information Systems Group
University of Kaiserslautern
Supervisor: your advisor

Zusammenfassung Das Testen von Datenbanken innerhalb einer Testumgebung kann nicht wirklich aussagekräftige Aussagen über das echte System geben, ohne eine exakte Kopie dessen zu besitzen. Um solche Kopien nicht nur exakt, sondern auch möglichst schnell zu erstellen, reichen die Möglichkeiten eines DB2-Administrators nicht aus, da ein Klonen eines DB2-Subsystems sehr komplex ist. Es werden Programme für den Mainframe benötigt, die fehlerfrei, schnell und so weit wie möglich automatisiert ablaufen können, um das Produktivsystem nicht zu schaden. Im Laufe der Seminararbeit wird am Beispiel des Instant CloningExpert for DB2 z/OS eine Möeglichkeit vorgestellt, wie ein solches System aufgebaut sein muss und wie es arbeitet.

1 Einführung - Warum Klonen?

Im Allgemeinen bedeutet Klonen eine Anfertigung einer 1:1-Kopie von Etwas, in unserem Falle einer großen Menge von Daten in einer DB2-Datenbank auf z/OS. Man könnte sich die Frage stellen, wieso man auf einem Mainframe überhaupt eine Kopie der Daten erzeugen lassen sollte, wenn doch sowieso alle Daten redundant gespeichert sind und ein Datenverlust sowohl bei Stromausfällen oder Hardwaredefekten als auch bei Katastrophen nahezu ausgeschlossen ist? Wie wir in den nächsten Abschnitten erfahren werden, ist Klonen mehr als einfach nur die Anfertigung eines Backups. Aus diesem Grund gibt es eine Reihe von Tools, die das Klonen von Daten unter z/OS ermöglichen und erleichtern.

Klonen wir eine Menge von Daten, bedeutet dies, dass wir einen Snapshot haben, den wir jederzeit wieder einspielen können, um eine Datenbank in genau diesen Zustand wieder zu versetzen. Dabei ist es egal, ob die Daten in das ursprüngliche oder in ein neues System eingespielt werden. Wird das Einspielen auf einem anderen System ausgeführt, ist es möglich, auf diesem System, welches das ursprüngliche nicht berührt, eine Kopie der Daten zu besitzen, um dort Tests durchzuführen. Da das ganze auf einer Kopie passiert, können getrost Änderungen vorgenommen werden. Das Testen von neuer Software ist somit zum einen ohne Risiko möglich, zum anderen kann die neue Software reelle und

sinnvolle Daten als Eingabe nutzen. Dies erspart die Eingabe von Testdaten und erprobt bereits das Verhalten im Produktivsystem.

Das gleiche gilt bei der Weiterentwicklung der eigenen Software. Es wäre leichtsinnig, eine neue Software-Version, welche unsere Daten verwendet, direkt auf dem Produktivsystem zu testen. Auch hier sollte zunächst eine 1:1 Kopie der Daten erstellt werden, die von der neuen Version genutzt werden kann. Eventuelle Fehler haben dann keine Auswirkungen auf die Original-Daten. Erst wenn die Version ausreichend und mit Erfolg getestet wurde, kann sie in das Produktivsystem integriert werden und die echten Daten verwenden.

Ein weiterer Einsatzzweck für eine Kopie auf einem separatem System ist das Generieren von Reporten und Statistiken über die Daten. Da dieser Vorgang meist die kompletten Daten berührt und aufwändige Rechenoperationen ausführt, sind lang andauernde Lesesperren auf den Daten notwendig. Im Produktivsystem würde dies zur Verlangsamung von Anfragen jeglicher Art führen und so den regulären Betrieb aufhalten.

Sollen die Daten für Schulungszwecke verwendet werden, beispielsweise um Personen die Möglichkeit zu geben, sich die Datenstrukturen anzuschauen und Beispielanfragen auszuprobieren, bevor diese Personen mit dem Produktivsystem arbeiten, kann auch hier eine geklonte Version der Datenbank verwendet werden. Wie bei den anderen Einsatzzwecken ist auch hier der Vorteil, dass das Verändern von Daten keine Auswirkungen auf das Produktivsystem hat und dass dessen Performance nicht leidet.

Es sind eine Vielzahl von Tools zum schnellen Klonen vorhanden. InfoHCopy von Infodesign [3] kann im Offline- und Onlinebetrieb ausgeführt werden. Ein anderes System ist der Instant CloningExpert for DB/2 z/OS, welches im Rahmen dieser Arbeit näher betrachtet wird.

2 Duplikation von Subsystemen

Wird ein System dupliziert, sprechen wir von einer homogenen Systemkopie, wenn auf dem Quell- und Zielsystem die gleiche Datenbankversion unter dem gleichen Betriebssystem läuft, beispielsweise in beiden Fällen DB2 z/OS. Komplizierter ist der Fall der heterogenen Systemkopie, was bedeutet, dass die Versionen auf den Systemen unterschiedlich sind.

Die in den folgenden Abschnitten vorgestellten Verfahren und Tools dienen zum Klonen von Datenbanken. Das heißt, nicht die kompletten Daten eines Systems einschließlich des z/OS Subsystems, Konfigurationen etc. werden kopiert, sondern nur die reinen zur Datenbank gehörenden Daten. Dies sind zum einen Definition der Datenbankobjekte sowie die Daten an sich (DB-Dump). Zum anderen werden DSNZPARM, BSDS und Logs beachtet, um nach dem Klonen der

Kataloge, Indexspaces und Tablespaces die geklonte Datenbank wieder in den Ursprungszustand versetzen zu können.

Es gibt drei Möglichkeiten, die Daten vom Quell- zum Zielsystem zu kopieren: Mittels DB-Dump, als temporärer Spiegelserver oder mit einem Disk-Dump. Beim Instant CloningExpert for DB2 z/OS funktioniert die erste Möglichkeit so, dass mit dem Tool ADRDSSU ein komplettes DB-Abbild erstellt wird, welches in das Zielsystem geladen werden kann. Dies beinhaltet alle Datenbankobjekte wie Datenbanken, Tabellen, Indexe, Trigger usw., sowie alle Daten.

Die zweite Möglichkeit nutzt die Spiegelserver-Funktionalität des Datenbanksystems aus. Richtet man einen neuen Spiegel ein und trennt danach wieder die Verbindung, besitzt dieser Spiegel den Datenbankzustand vom Zeitpunkt der Trennung.
Die dritte Methode funktioniert direkt auf Festplattenebene: Die Daten werden komplett und 1:1 von der Festplatte des Quellsystems auf die Festplatte des Zielsystems kopiert.

Damit die Kopie in Betrieb gehen kann, sind noch einige Änderungen nötig. Zum einen kann ein Umbenennen der neuen Datenbank sinnvoll sein, vor allem wenn Quell- und Zielserver nicht isoliert von einander - oder sogar identisch - sind. Weitere Schritte sind das Einbringen von gesammelten Log-Dateien (Active Logs, Archive Logs, ...) in die neue Datenbank, damit diese den aktuellen und konsistenten Zustand erreicht. Anschließend kann das Zielsystem gestartet und, nach kurzer Anpassung der DB2-Einstellungen, verwendet werden.

2.1 Ziele

Im vorherigen Abschnitt haben wir erfahren, wie Klonen funktioniert: Kopie anfertigen, ins Zielsystem einspielen, noch etwas anpassen und fertig! Betrachten wir jedoch mal folgende Punkte, fällt uns schnell auf, dass Klonen gar nicht so einfach ist, wie es sich anhört:

Das Lesen der kompletten Daten dauert sehr lange und benötigt viele Ressourcen sowie Lesesperren. Am schnellsten ginge es, wenn das Quellsystem vor dem Klonen beendet wird. Dann kann der Klon-Prozess alle Ressourcen exklusiv nutzen. Weder die laufenden Aktivitäten auf der Datenbank beeinflussen das Klonen noch beeinflusst das Klonen die laufenden Aktivitäten. Oft kommt diese Offline-Variante jedoch gar nicht erst in Frage, da das System nicht beendet werden darf. Denn dies widerspricht dem Zero-Downtime-Gedanken (bzw. der erwünschten 99,999% Verfügbarkeit) des Mainframes.

Möchte man online klonen, lässt es sich nicht vermeiden, dass dies die Performance der anderen Prozesse verschlechtern wird. Also gilt es zu überlegen, wie das Klonen am wenigsten Ressourcen belegt. Weiterhin ist es nützlich, das Klonen so schnell es geht passieren zu lassen. Ist das Klonen nämlich beendet,

stehen den restlichen Prozessen wieder einhundert Prozent der Leistung zu.
Bei der Betrachtung der drei Möglichkeiten aus dem vorherigen Abschnitt, wie eine Kopie angelegt werden kann, fällt schnell auf, dass die direkte Disk-Kopie die günstigste ist. Während das Erstellen eines DB-Dumps bzw. eines Spiegelservers viel Rechenleistung und Arbeitsspeicher benötigt und die Daten auf der Platte eventuell mehrfach gelesen werden müssen, wird die Festplatte beim Erstellen des Disk-Dump nur ein einziges mal von Beginn bis zum Ende gelesen. Da Spiegelserver eigentlich nicht für das Anfertigen von Klonen, sondern auf Hochverfügbarkeit und Lastverteilung optimiert sind, dauert auch diese Methode länger als die direkte Disk-Kopie. Deshalb ist die Disk-Kopie die schnellste Art zu Klonen.

Neben Schnelligkeit, Ressourceneffizienz und der Möglichkeit, das Klonen online durchzuführen, ist als weiteres wichtiges Ziel die Datenkonsistenz zu nennen. Ein Cloning-Tool muss dafür sorgen, dass die kopierten Daten im Zielsystem in einem konsistenten Zustand vorliegen werden. Andernfalls wäre der Klon für Tests, Reports und Backups unbrauchbar.

2.2 Hinweise

Beim Klonen von RACF-Definitionen, welche den Zweck haben, Ressourcen durch Verwaltung von Zugriffsrechten zu schützen, muss darauf geachtet werden, dass diese an das Zielsystem angepasst werden. Weiterhin müssen Änderungen beachtet werden, etwa falls der Subsystem-Typ von data sharing auf non-data sharing geändert werden soll. Jegliche Art weiterer Veränderungen und Besonderheiten erfordern noch höhere Beachtung. Dazu zählt beispielsweise auch eine Datenbankkopie in eine andere DB2-Version. Je verschiedener Quell- und Zielsystem sind und je mehr weitere Besonderheiten auftreten, desto komplexer wird der Klon-Vorgang.

2.3 Unterstützung durch Tools

Es gibt eine Reihe von Tools, die dem Benutzer das Klonen vereinfachen. So nehmen sie ihm einen Großteil der Arbeit ab, vereinfachen die Bedienung des Klonens und können effizienter arbeiten. Mit der Benutzung eines solchen Tools passiert das Klonen also fast wie von selbst. Zusätzlich kann das Tool Speicher- und DB2-Funktionen optimal ausnutzen und somit den Vorgang enorm beschleunigen. Außerdem beachten die Tools die Besonderheiten aus dem vorherigen Abschnitt, also etwa wenn zwei unterschiedliche DB2-Versionen verwendet werden.

Die HSC-Komponente (Homogeneous System Copies) von Instant Cloning-Expert ermöglicht effiziente heterogene Systemkopien. Der Benutzer wird Schritt für Schritt durch den Klon-Assistenten geführt, um alle Einstellungen vorzunehmen. Der Rest regelt HSC automatisch. Die HSC-Komponente arbeitet wie folgt:

– Der Performance und Geschwindigkeit wegen wird die direkte Kopie auf Plattenebene (Instant Copy) durchgeführt.

– Datasets werden mittels schnellem „Low-Level-Rename“ umbenannt.
– Der Klon-Prozess wird Schritt für Schritt detailliert erklärt und verifiziert, um unbeabsichtigte Einstellungen zu vermeiden.
– Es wird sich intensiv um Besonderheiten gekümmert, wie z.B. der Wechsel von data sharing auf non-data sharing.
– Zum Schluss überprüft das Tool die erstellte Kopie.

3 Mergen / Duplizieren

Zwei verschiedene Möglichkeiten des Klonens sind das Mergen und Duplizieren. Beim Mergen werden verschiedene Datenbanken (z.B. DB2-A, DB2-B, DB2-C) in eine Datenbank DB2-N zusammengefasst. Beim Duplizieren wird etwas anderes erreicht: Aus einer einzelnen DB2-Datenbank werden gleichartige Klone dupliziert.

3.1 Einzelheiten

Beim Klonen werden alle oder nur ein Teil der Daten des Systems dupliziert. Dabei können Quelle und Ziel verschieden oder auch gleich sein, da es auch möglich ist Objekte in ein und dasselbe DB2-System zu duplizieren. In dem vorgestellten Instant Cloning Tool kann dies vorher angegeben werden und so genauer spezifiziert werden, welche Teile der Datenbank geklont werden sollen. Ist das Zielsystem das Quellsystem selbst, erhalten existierende Objekte allerdings nur einen „Refresh“. Dieser „Refresh“ wird immer dann verwendet, wenn entsprechende Objekte schon vorhanden sind und entspricht einer Aktualisierung der enthaltenen Daten.

Geklont werden hierbei ausschließlich Datenbankobjekte, was heißt, dass nicht notwendigerweise auch Teile des DB2-Subsystems geklont werden, wie z.B. Katalog und Directory oder andere DB2-Subsystem spezifische Teile. Dies ist meistens auch gar nicht erwünscht, wenn zum Beispiel nur ein Testsystem angelegt werden soll.

Um ein geklontes System auf den aktuellen Stand des Originalsystems zu bringen, werden einige Schritte benötigt, die beim Klonen implementiert werden müssen:

– Objektbereich definieren: Hier wird angegeben, welche Datenbankobjekte genau geklont werden sollen
– Abhängige Objekte bestimmen: Nachdem der Objektbereich definiert wurde, werden anschließend die von diesen Objekten abhängigen Objekte identifiziert. Dazu gehören zum Beispiel Indexe und Views, aber auch Authorisierungsobjekte, falls diese benötigt werden
– DDL extrahieren: Anhand der angegebenen Objekte werden anschließend die Definitionen für die neuen Objekte anhand von DDL erzeugt

– Daten extrahieren: Anhand der angegebenen Objekte werden die Daten dieser Objekte ausgelesen
– Eventuelles umbenennen: Falls gewünscht werden Namensänderungen durchgeführt
– DDL ausführen: Nun wird das erzeugte DDL auf dem Zielsystem ausgeführt und so die für das Klonen benötigten Objekte erstellt
– Daten einlesen: Nach dem Erstellen der Objekte auf dem Zielsystem werden anschließend die extrahierten Daten in das Zielsystem geladen

Eine Ausnahme bildet hierbei der schon angesprochene „Refresh“. Dieser funktioniert genauso, nur werden hier ausschließlich die Daten geklont und nicht die Datenbankobjekte, da diese ja schon vorhanden sind.

3.2 Eigenschaften

Die Ziele eines solchen Klon-Verfahrens sind das Sparen von Zeit und Ressourcen. Das kann durch das Erstellen so genannter Instant Copies erreicht werden, da diese sehr schnell sind und kaum CPU benötigen. Allerdings gibt es beim Klonen dieser Art auch Probleme:

– Wie werden Sequenzen richtig behandelt?
– XML wird von DSN1COPY [4] nicht unterstützt
– Ist die Verfügbarkeit des Quellsystems garantiert? (Kurze Ausfälle von Datenbanksystemen können sehr viel Geld kosten)
– Werden User Defined Objects richtig behandelt?

3.3 Unterstützung des Klones mit dem Instant CloningExpert for DB2 z/OS

Um diese Probleme richtig zu behandeln benötigt man ein Tool, das solche Probleme möglichst verhindert und einfach zu benutzen ist. Weiterhin sollte es flexibel sein, einen hohen Grad an Automatisierung besitzen und sehr effizient sein.

Hier bietet wieder der angesprochene Instant CloningExpert for DB2 z/OS eine Möglichkeit an, um das Mergen und Duplizieren zu unterstützen. Die Komponente des Instant CloningExpert for DB2 z/OS, die dafür zuständig ist, wird HOC genannt.

HOC kann DDL mittels sehr schnellem DSNTIAD [6] (DSNTIAD ist ein in Assembler geschriebenes Programm und bietet die Möglichkeit dynamische SQL-Statements zu erzeugen) bearbeiten. Weiterhin bietet HOC ein schnelles und flexibles Umbenennen von Datasets mit Wildcard Support.

HOC kann auch mit komplexen Abhängigkeiten und Strukturen umgehen. Diese Unterstützung ist allerdings optional. Dafür erkennt es Sequenzen und

versucht diese richtig zu behandeln.

Zusammenfassend bietet die HOC-Komponente des Instant CloningExpert for DB2 z/OS mit DB2 COPIES und DSN1COPY (DSN1COPY ist ein Utility von IBM mit dem DB2 VSAM Data Sets geklont werden können) und der Kompabilität bis DB2 Version 10 eine sehr gute Lösung Mergen oder Duplizieren von DB2-Systemen zu unterstützen.

Dabei geht HOC ähnlich den oben genannten Schritten vor. Allerdings unterscheidet die Komponente Objektklonen von Datenklonen. Deshalb werden in HOC die Schritte in zwei Bereiche aufgeteilt:

1. Objektklonen wird durchgeführt, indem zuerst die Objektdaten von der Quelle extrahiert werden, um anschließend ein DDL zu erzeugen. Dieses DDL wird dann auf dem Ziel ausgeführt, um somit wieder die entsprechenden (ausgesuchten) Objekte zu erzeugen.
2. Datenklonen wird durchgeführt, indem DSN1COPY benutzt wird, um Kopien oder VSAM DB2 Cluster zu klonen

Um eine hohe Verfügbarkeit zu garantieren wird so nur ein „Refresh“ mittels DB2 Clone Tables durchgeführt, da die entsprechenden Objekte schon angelegt sind. (Im Gegensatz zu der vorher vorgestellten Reihenfolge)

3.4 Weitere Tools

Natürlich existiert nicht nur ein Tool, mit dem das Klonen auf z/OS Datenbanken möglich ist. Zum Duplizieren von SAP R/3 Systemen mit DB2 auf z/OS, kann zum Beispiel die Software InfoHcopy von InfoDesign Software helfen [3].

Diese wird häufig eingesetzt, um aus einem Produktionssystem ein Qualitäts- oder Testsystem aufzubauen. ähnlich wie bei der HOC-Komponente des Instant CloningExpert for DB2 z/OS, werden auch hier auf Kopien basierend Klone in mehreren Arbeitsschritten erzeugt. Dieses Verfahren kann offline oder auch im parallelen Online-Betrieb durchgeführt werden, so dass die Quelle hochverfügbar bleiben kann.

Ein weiteres Tool ist zum Beispiel EMC ResourcePak Extended für z/OS. Dies ist ein Paket von nützlichen Dienstprogrammen speziell für Mainframes. Dafür werden die Datenbankobjekte mit EMC TimeFinder [1] [2] geklont und die Daten anschließend mit den Programmen des EMC ResourcePak Extended für z/OS synchronisiert. So können auch automatisierte Klonvorgänge erzeugt und implementiert werden.

EMC TimeFinder ist selbst wieder eine Produktfamilie, die im Großen und Ganzen aus zwei unterschiedlichen Produkten besteht:

1. TimeFinder/Clone: Ermöglicht es, Kopien von Daten auf mehrere Zielsysteme zu klonen. Dabei dürfen die Daten von einer Quelle in bis zu 16 Ziele geklont werden.
2. TimeFinder/Snap: Ermöglicht es Benutzern, spezielle virtuelle Devices zu erstellen. Hier kann im Gegensatz zu TimeFinder/Clone in bis zu 128 virtuelle Ziele geklont werden.

Wie auch InfoHcopy, kann EMC TimeFinder sowohl online, als auch offline für das Klonen benutzt werden.

Der CA Mainframe Configuration Manager [8] von CA Technologies besitzt ein Batch-Utility zum Mergen/Klonen, indem IMS SYSGEN Sources (Mit der so genannten IMS SYSGEN Source werden verschiedene Makros angesprochen, die helfen ein IMS-System zu konfigurieren) erstellt wird, die zum Mergen mehrerer IMS-Systeme (IMS ist ein Informationssystem von IBM und besitzt verschiedene Schnittstellen und Komponenten) oder zum Klonen eines existierenden IMS-Systems benutzt werden können. Dieses Batch-Utility versucht entstehende Konflikte zu lösen und bei der Erstellung eines Sysplexes zu helfen.

Auch von Rocketsoftware existieren einige Programme die das Klonen von DB2-Datenbanken auf einem Mainframe unterstützen [7]:

1. Mainstar Rapid Database Refresh for IMS on z/OS: Klont IMS-Datenbanken
2. Mainstar VCR for DB2 on z/OS: Klont DB2-Subsysteme auf Volume Level
3. Mainstar Fast Table Space Refresh for DB2 on z/OS: Klont DB2-Tablespaces und Indexspaces

4 Zusammenfassung

Der Instant CloningExpert for DB2 z/OS besteht also im Großen und Ganzen aus zwei Komponenten:

1. Die HSC Komponente, die für die Duplikation eines Subsystems zuständig ist (Homogene Systemkopie)
2. Die HOC Komponente, die für das Mergen und Duplizieren eines Systems und seiner Daten zuständig ist. Diese kann ein ganzes System oder Teile dessen erzeugen (Homogene Objektkopie)

Mit dem Instant CloningExpert for DB2 z/OS ist es also möglich, mehrere unabhängige DB2-Systeme in eine so genannte Data Sharing Group zusammenzufügen, indem eine mit der HSC-Komponente geklont wird und die anderen mittels der HOC-Komponente diese neue Data Sharing Group “refreshen„ können.

Diese Komponenten sind flexibel, schnell und einfach zu benutzen. Somit können Fehler eines z/OS-Administrators vermieden werden und verschiedenste Klon-Möglichkeiten verwendet werden. Weiterhin kann durch die starke Automatisierung die Hochverfügbarkeit des Systems gesichert werden und die komplexen Vorgänge sind weniger fehleranfällig.

Glossar

ADRDSSU	Programm zum direkten Speicherzugriff und Speichermanagement
BSDS	Bootstrap Data Set
DDL	Data Definition Language
DSN1COPY	Programm zum Kopieren von VSAM-Datasets
DSNTIAD	Programm zum dynamischen Ausführen von SQL-Updates
DSNZPARM	DB2-Subsystem-Konfigurationsparameter
HOC	Homogeneous Object Cloning
HSC	Homogeneous System Copies
IMS	Information Management System
RACF	Resource Access Control Facility
VSAM	Virtual Storage Access Method

Literatur

1. EMC Corporation. Ihr 15 Minuten-Leitfaden zum Thema Mainframe-Umgebungen, 2008.
2. EMC Corporation. Using EMC TimeFinder to Clone DB2 for Linux, UNIX, and Windows Databases, 2010.
3. InfoDesign GmbH. InfoDesign Software - InfoHcopy, SAP R/3 mit DB2 auf IBM S/390 und z/OS, Homogene Systemkopien- online- ohne Produktionsunterbrechung. `http://www.infodesigner.biz/pdfs/Homepage-Flyer_Homogene_short_12_2003.pdf`, 2004.
4. IBM. DB2 Version 9.1 for z/OS - DSN1COPY. `http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.db29.doc.ugref/db2z_utl_dsn1copy.htm`, 2010.
5. SEGUS Inc. and SOFTWARE ENGINEERING GMBH. Cloning - What's new and faster? 2010.
6. Craig S. Mullins. DSNTIAD - The Dynamic SQL Update Program. `http://db2portal.blogspot.com/2006/03/dsntiad-dynamic-sql-update-program.html`, 2006.
7. Rocket Software. Rocket Mainstar - Optimizing System z. 2010.
8. CA Technologies. CA Mainframe Configuration Manager for IMS for z/OS. `http://www.ca.com/files/ProductBriefs/2433_mainframeconfig_ps_v2_0710_241423.pdf`, 2010.