Eine parametrisierbare Messumgebung für datenintensive verteilte Anwendungen

Joachim Klein, Theo Härder

TU Kaiserslautern, Postfach 3049, 67653 Kaiserslautern
{jklein,haerder}@informatik.uni-kl.de

Abstract

Datenintensive Anwendungen werden heute in der Regel verteilt über mehrere horizontal oder vertikal verknüpfte Rechnerknoten abgewickelt. Dabei ist das Leistungsverhalten des Gesamtsystems seit Jahrzehnten, das *immerwährende* Problem, da den technologischen und algorithmischen Verbesserungen stets zunehmende Anforderungen der Anwendungen auf noch schneller wachsenden Datenvolumina gegenüberstanden. In dieser Zeit bildeten sich auch verschiedenartige Architekturformen für die Datenhaltung heraus, deren Performanz bei Entwicklung und Einsatz zu bestimmen und zu optimieren war. Aus diesen Gründen haben wir ein Framework entwickelt, das ein neues Konzept zur Leistungsmessung verkörpert und das sich durch geeignete Abstraktionen in einfacher Weise an viele unterschiedliche Systemumgebungen anpassen lässt. In diesem Beitrag stellen wir die Entwurfskonzepte für dieses System zur Unterstützung der systematischen Leistungsanalyse datenintensiver Anwendungen vor und zeigen seinen Einsatz und seine Parametrisierbarkeit am Beispiel des Datenbank-Caching in Web-basierten Client/Server-Anwendungen.

1 Motivation

Verteilte Systeme und verteilte Anwendungen sind schon seit längerer Zeit Teil unserer täglichen Arbeit am Computer, z. B. beim Schreiben einer E-Mail oder beim Durchsuchen eines Online-Angebots. Für die Zukunft ist zu erwarten, dass sich ihr Einsatz durch die stets größer werdende Vernetzung und den rasch wachsenden Markt für mobile Endgeräte noch deutlich steigert. Vor allem datenintensive Anwendungen können heutzutage ihren hohen Anforderungen (z. B. an die Antwortzeit) oft nur noch durch eine Verteilung der Lasten gerecht werden. In der Forschung und Entwicklung dieser Anwendungen ist die Erfassung von Leistungsdaten ein sehr wichtiger Entscheidungsfaktor. Die erfassten Werte dienen nicht nur zur Leistungsanalyse und Optimierung, sondern auch zum Verfolgen der verteilten Abläufe oder zur Fehleranalyse. Die Vermessung verteilter Anwendungen wird jedoch oft – durch eine große Heterogenität der eingesetzten Teilanwendungen, die zu einem Gesamtsystem zusammengeschlossen wurden – behindert. Jede Applikation bietet eigene Schnittstellen und nicht immer lassen sie sich zur besseren Beobachtung individuell anpassen.

Aus diesen Gründen haben wir versucht, die Vermessung datenintensiver Anwendungen durch ein eigenständiges Framework bestmöglich zu unterstützen. Es ermöglicht durch die Definition geeigneter Schnittstellen, die Struktur des verteilten Systems genau zu erfassen, und darüber alle für die Messung notwendigen Informationen. Darüber hinaus wurde die Möglichkeit vorgesehen, definierte Messläufe mehrfach und möglichst automatisiert durchzuführen, was die Konfiguration und Wartung der Teilanwendungen zwischen den einzelnen Messläufen mit einschließt. Für stets wiederkehrend anfallende Aufgaben, wie z. B. die Übertragung der Messdaten und deren Speicherung, bietet die Messumgebung generisch einsetzbare Implementierungen an, die sich vielfältig erweitern lassen.

Im folgenden Abschnitt werden die wichtigsten Konzepte und Lösungsansätze zur Unterstützung verteilter Leistungsanalysen kurz beschrieben. Abschnitt 3 (Einsatz) zeigt den Einsatz anhand eines praktischen Beispiels im Bereich Datenbank-Caching [5].

2 Konzepte

Um die eingesetzten Konzepte besser in einen Gesamtzusammenhang bringen zu können, betrachten wir zunächst den stark vereinfachten Grundaufbau eines verteilten Messvorgangs (vgl. Abb. 1). An diesem diskutieren wir die verwendeten Begriffe und die wichtigsten Aufgaben der Messumgebung.

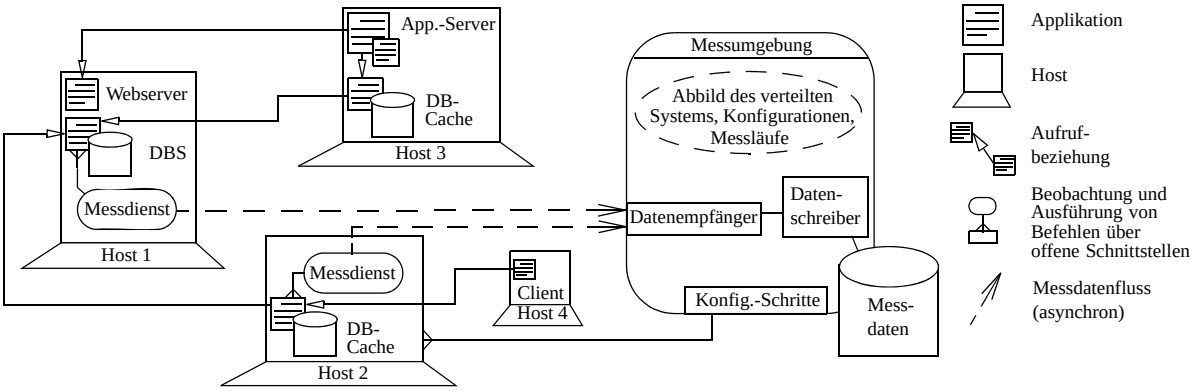

Abb. 1: Grundaufbau einer verteilten Messung mit Hilfe der entwickelten Messumgebung.

Jedes zu vermessende, verteilte System besteht aus Teilanwendungen, die untereinander kommunizieren. Die einfachste und häufigste Art der Kommunikation ist ein Funktionsaufruf über eine zur Verfügung gestellte Schnittstelle (z. B. JDBC-Schnittstelle eines DBS). Der Messumgebung werden zunächst die Applikationen bekannt gemacht, welche während eines Messlaufes überwacht oder angepasst werden müssen. Eine solche Applikation (Knoten) des verteilten Systems nennen wir im Kontext des Mess-Frameworks *Arbeitsknoten*. Jeder Arbeitsknoten verfügt über eine Reihe obligatorischer sowie optionaler Eigenschaften, die für einen Messlauf deklarierbar sind. Im Folgenden sind die wichtigsten Eigenschaften kurz dargestellt:

- **Aufrufbarkeit**
  Jeder Arbeitsknoten verfügt über mindestens eine Schnittstelle, über die er aufgerufen werden kann. Die Kontextinformationen werden von der Messumgebung über sogenannte *Service-Kontexte* erfasst.
- **Aufrufbeziehungen zu anderen Arbeitsknoten**
  Für jeden Arbeitsknoten lässt sich angeben, auf welche anderen er zugreift.
- **Beobachtbarkeit und Messdatenlieferant**
  Da ein Arbeitsknoten eine Applikation repräsentiert, besteht grundsätzlich die Möglichkeit diese zu beobachten. Die Beobachtungspotenziale hängen jedoch stark von den Eingriffsmöglichkeiten in die Anwendung ab. Ist eine zu beobachtende Applikation z. B. eine eigene Entwicklung und somit frei erweiterbar, können alle benötigten Schnittstellen und Überwachungsfunktionalitäten der Anwendung hinzugefügt werden. Die Applikation ist somit *komplett beobachtbar*. Sind die Quellen einer Anwendung nicht anpassbar, so verfügt sie evtl. über Beobachtungsschnittstellen, die aufgerufen und abgefragt werden können (z. B. ein Datenbanksystem). Die Anwendung ist somit *teilweise beobachtbar*. Ist die Anwendung aufgrund fehlender Schnittstellen *nicht beobachtbar*, so können für diese immer noch Messwerte erfasst werden, z. B. durch Abfrage von Leistungsdaten über die Ablaufumgebung oder durch das von außen beobachtbare Verhalten (black-box testing).
- **Konfigurierbarkeit**
  Alle Arbeitsknoten sind konfigurierbar, da sie zumindest gestartet und beendet werden können. Es besteht jedoch meist die Möglichkeit über Konfigurationsdateien oder Management-Schnittstellen die Anwendung zu kontrollieren. Das Mess-Framework definiert daher *Konfigurierungsschritte*, die automatisch vor einem Messlauf ausgeführt werden können (z. B. Leeren des Cache-Speichers). Das Framework kann über einen Konfigurator, der eine Menge von Schritten sammelt, sicherstellen, dass diese gemeinsam und in der richtigen Reihenfolge ausgeführt werden.

Ein spezieller, vom Mess-Framework zur Verfügung gestellter Arbeitsknoten ist der sogenannte simulierte Client-Arbeitsknoten, der zusätzlich die Möglichkeit anbietet, die Arbeitslast einer Client-Anwendung zu emulieren. Hierzu bedient er sich eines Schedulers, der vorab definierte Arbeitsschritte in vorgegebener relativer Häufigkeit (im Bezug zu anderen Arbeitsschritten) oder Wiederholungsanzahl ausführt. Genauere Informationen über den in das Framework integrierten Scheduler findet man in [7].

Die wichtigste Aufgabe des Mess-Frameworks ist die *Messdatenverwaltung*. Um die Erfassung und Speicherung der Messdaten bestmöglich zu unterstützen, wird vorab eine Modellierung der Werte vorgenommen, die auf einem Arbeitsknoten erfassbar sind. Zu diesem Zweck verwendet das Framework

das Konzept der *Ausführungskontexte*. Ein Ausführungskontext (kurz: Kontext) repräsentiert die Bearbeitung einer bestimmten Aufgabe (z. B. die Funktionalität einer Klasse oder einer Funktion) in einer Teilanwendung. So kann z. B. ein Ausführungskontext „Query“ die Bearbeitungen im Kontext einer Anfrage beschreiben. Für jeden Ausführungskontext werden als Attribute die erfassbaren Messwerte definiert. Ein Messwert kann einfach oder mehrfach in der Instanz eines Ausführungskontextes erfasst werden. Damit auch Programmverzweigungen und Unteraufgaben abbildbar sind und mit einer Hauptaufgabe in Bezug gebracht werden können, erlaubt das Framework die Erzeugung von Unterkontexten. Der stets übergeordnete Wurzel-Kontext für alle Ausführungskontexte ist der *initiale Kontext*. An ihn können Messwerte, wie z. B. eine periodisch erfasste Speicherauslastung, gebunden werden, die bezüglich der kompletten Teilanwendung Gültigkeit besitzen. Abb. 2 zeigt beispielhaft die Modellierung der Ausführungskontexte für das in Abschnitt 3 (Einsatz) vermessene Datenbank-Cache-System. Es werden Messergebnisse für die Bearbeitung einer Anfrage (Query) erfasst, für welche jeweils eine Analyse durchzuführen ist und eventuell mehrere Nachladevorgänge ausgelöst werden.

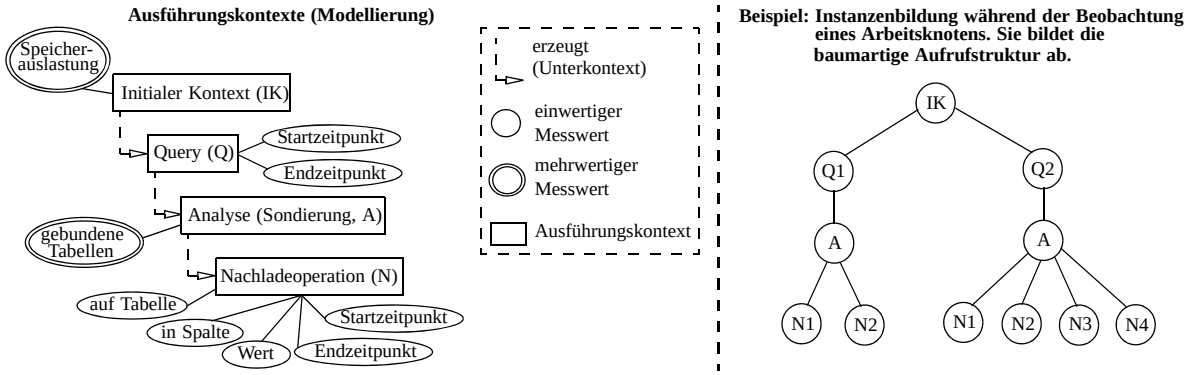

Abb. 2: Modellierung der Ausführungskontexte für einen Datenbank-Cache.

Um Beziehungen zwischen Ausführungskontexten verschiedener Teilanwendungen abzubilden, die z. B. bei einem entfernten Funktionsaufruf entstehen, besteht grundsätzlich die Möglichkeit die Referenz des aktuellen Ausführungskontextes zu übergeben. Besteht jedoch keine ausreichende Eingriffsmöglichkeit in die aufgerufene Applikation, kann eine direkte Übermittlung der Referenz nicht stattfinden. Ansätze, die Beziehungen zwischen Ausführungskontexten verschiedener Teilanwendungen trotzdem abzubilden, finden sich in [7].

Die vorangehend gezeigte Modellierung der Ausführungskontexte wird vom Framework genutzt, um die Messdatenerfassung und die Messdatenspeicherung deutlich zu vereinfachen. Die Erfassung von Messdaten wird parallel für jeden Arbeitsknoten durch sogenannte *Beobachter* erledigt. Damit diese „nahe“ der zu beobachtenden Anwendung ausgeführt werden können, bedient sich die Messumgebung eines „Gehilfen“, der *Messdienst* genannt wird. Jeder Arbeitsknoten kann diesen initialisieren, um eine Außenstelle auf einem entfernten Host einzurichten. Der Messdienst führt dort weitere Programme aus oder verarbeitet Befehle über die frei erweiterbare Kommandoschnittstelle des Dienstes (vgl. Abb. 1). Ein Beobachter registriert sich an der zu überwachenden Teilanwendung, die diesen unterrichtet, falls neue Messwerte vorliegen. Die erhaltenen Werte erfasst der Beobachter über die ihm zur Verfügung gestellte Implementierung der vorher modellierten Ausführungskontexte, welche durch einen im Framework verankerten Generator erzeugt wird, so dass Fehler vermieden und die modellierten Beziehungen zwischen den Kontexten eingehalten werden.

Die asynchrone Übertragung der auf diese Weise erfassten Messwerte wird durch das Framework übernommen und läuft für den Entwickler verborgen ab. Der Messdienst hat die Option, gesammelte Nachrichten kontinuierlich in Paketen bestimmter Größe oder insgesamt erst am Ende der Messung an die Messumgebung zu übertragen.

Nachdem die Messdaten an die Messumgebung übermittelt wurden, speichert ein Datenschreiber diese auf einen persistenten Speicher. Das Framework unterstützt zur Zeit die Sicherung der Daten in ein relationales Datenbanksystem. Es ist nicht notwendig, ein spezielles Schema für die Aufnahme der

Werte zu entwickeln. Das Framework verwendet abermals die Modellierung der Ausführungskontexte und erzeugt für diese geeignete Tabellen, in denen die Instanzen der Kontexte mit ihren Werten und Beziehungen gespeichert werden.

3 Einsatz

Das Mess-Framework wird zur Umsetzung von Messungen für das Constraint-basierte Datenbank-Caching (CBDC) [6] benutzt, das z. B. in Web-basierten Client/Server-Anwendungen eingesetzt werden kann [7]. Im Forschungsbereich Datenbank-Caching wurden bereits einige andere Ansätze beschrieben, mit denen sich der neue Ansatz vergleichen muss (z. B. DBProxy [2], MTCache [8] oder das Vorhalten von Replikaten zur Verbesserung einzelner Anfragen [1]).

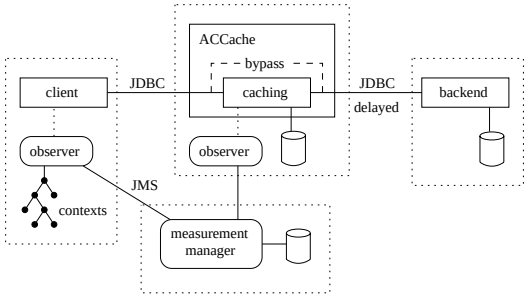

Abb. 3: Aufbau des vermessenen Systems.

Zur Durchführung mehrerer einfacher Messungen, basierend auf dem Prototyp des CBDC-Systems (ACCache) [4], wurde das in Abb. 3 gezeigte verteilte System aufgebaut und anschließend in der Messumgebung abgebildet. Auf der Backend-Datenbank wurden TPC-W-Daten zur Verfügung gestellt und für das ACCache-System eine geeignete Cache-Group-Definition[1] [3, 6] entwickelt. Für den simulierten Client-Arbeitsknoten (Client) und den ACCache-Arbeitsknoten wurden Beobachter implementiert. Durch eine entsprechende Modellierung der Ausführungskontexte wurde der Client-Beobachter in die Lage versetzt, mehrere Zeitpunkte während der Abarbeitung einer Anfrage zu erfassen. Auf dem ACCache-System wurden die in Abb. 2 gezeigten Messgrößen beobachtet. Dabei wurde eine auf dem ACCache abgewickelte Anfrage mit dem entsprechenden Anfragekontext auf dem Client durch Übergabe der Kontextreferenz verknüpft. So kann beim Einsatz mehrerer Clients stets entschieden werden, von wem die auf dem ACCache durchgeführte Bearbeitung ausgelöst wurde.

Zur Definition einer Arbeitslast wurde für den simulierten Client-Arbeitsknoten der Arbeitsschritt „order display“, in Anlehnung an das im TPC-W-Bechmark beschriebene Workload-Profil [9], entwickelt. Innerhalb des Arbeitsschritts werden die Anfragen „Order“ (O), „OrderLine“ (OL) und „Item“ (I) ausgeführt. Dabei fragt „Order“ die Informationen einer Bestellung und „OrderLine“ die zugehörigen Positionsinformationen des Auftrages ab. Die „Item“-Anfrage gibt Detailinformationen zu einer zuvor ausgelesenen Bestellposition zurück. Der angelegte Arbeitsschritt wurde durch den Scheduler mehrfach hintereinander ausgeführt, um den Caching-Effekt beobachten zu können. In den Messläufen wurde die Latenz zum Backend (0 ms, 20 ms, 50 ms) variiert, wobei die Überbrückung des Caches (Bypass-Option, vgl. Abb. 3) jeweils einmal ein- und ausgeschaltet wurde. Alle Läufe wurden zur Absicherung der Ergebnisse dreifach wiederholt. Abb. 4 illustriert beispielhaft einige Ergebnisse der Messläufe,

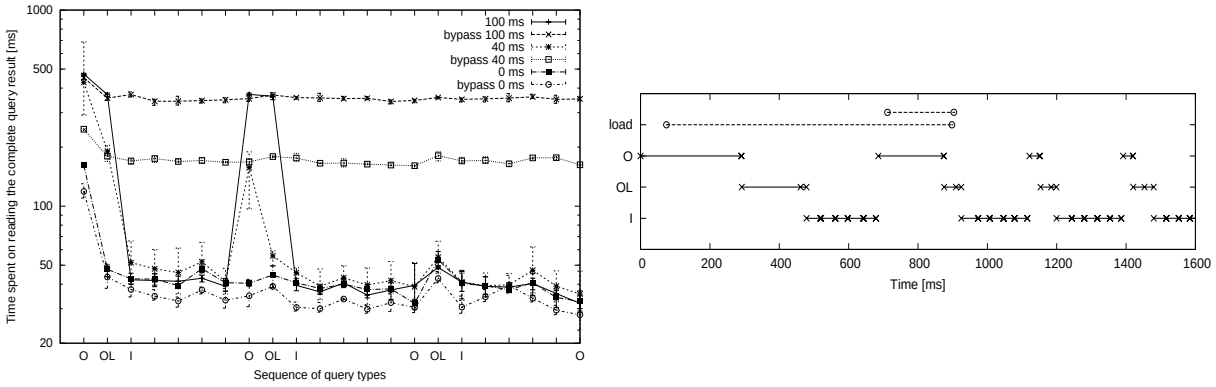

Abb. 4: Mit der Messumgebung erhobene Leistungsdaten.

[1] Eine Cache-Group definiert die im Cache gehaltenen Tabellen und zusätzliche Constraints, um zu entscheiden, welche Tupel in den Cache zu laden sind.

wobei links die Bearbeitungszeiten und rechts die zeitlichen Abläufe der Anfragen dargestellt sind. Die rechte Auswertung, bei der die Latenz zum Backend 20 ms betrug, zeigt zusätzlich die Zeitspanne, in der das Nachladen stattfand.

Damit die zuvor beschriebenen Messläufe nacheinander – in den verschiedenen Konfigurationen und ohne menschliche Eingriffe – durchgeführt werden konnten, mussten verschiedene Konfigurierungsschritte entwickelt werden. Diese umfassen allgemeine Aufgaben, wie z. B. das Starten des Messdienstes auf einem entfernten Host, und arbeitsknotenspezifische Aufgaben, wie z. B. das Leeren des Cache-Inhaltes.

4 Zusammenfassung

Die vorgestellten Konzepte und Komponenten des entwickelten Mess-Frameworks, die nicht auf eine bestimmte Rechner- bzw. Programmarchitektur festgelegt sind, bilden eine umfassende Basis, um verteilte Messungen durchzuführen. Die Beschreibung einzelner Teilanwendungen durch Arbeitsknoten ermöglicht es, die Struktur des verteilten Systems genau zu erfassen und alle Konfigurationsinformationen klar zu strukturieren. Der simulierte Client-Arbeitsknoten emuliert die Arbeitslast einer Client-Anwendung und ermöglicht die Generierung beliebiger Lastprofile. Allgemeingültige und arbeitsknotenspezifische Konfigurierungsschritte erweitern, ähnlich wie eine Plugin-Schnittstelle, den Funktionsumfang der Messumgebung und ermöglichen dadurch die automatische Verwaltung immer komplexerer Systemkonfigurationen.

Durch die komfortable und einfache Modellierung von Ausführungskontexten wird die Messdatenerfassung und -speicherung fast vollständig automatisiert. Ist ein freier Eingriff in die Funktionalität einer Teilanwendung gegeben, können die vorgenommenen Modellierungen auch für die Entwicklung einer noch nicht vorhandenen Monitoring-Schnittstelle herangezogen werden.

Der erste Einsatz des Frameworks hat gezeigt, dass eine rasche Umsetzung der Konzepte auf ein konkretes System möglich ist. Durch den Einsatz des Messdienstes und die asynchrone Übertragung der Messdaten wurde der Einfluss der Beobachtung auf die einzelnen Teilanwendungen möglichst gering gehalten.

Das Framework stellt bislang jedoch keine Möglichkeit zur Verfügung, die Auswertung der erfassten Ergebnisse zu automatisieren. Eine automatische Verdichtung von Werten oder sogar das Auffinden von Ausreißerwerten für Ausführungskontexte, könnten sinnvolle Erweiterungen darstellen.

Literatur

[1] M. Altinel, C. Bornhövd, S. Krishnamurthy, C. Mohan, H. Pirahesh, B. Reinwald: Cache Tables: Paving the Way for an Adaptive Database Cache. VLDB Conference 2003: 718–729

[2] K. Amiri, S. Park, R. Tewari, S. Padmanabhan: DBProxy: A Dynamic Data Cache for Web Applications. ICDE Conference 2003: 821–831

[3] A. Bühmann, J. Klein: Examining the Performance of a Constraint-Based Database Cache, Submitted to: ADBIS, September 29 - October 03, 2007, Bulgaria, URL: http://wwwdvs.informatik.uni-kl.de/pubs/papers/BK07.ADBIS.html

[4] A. Bühmann, T. Härder, C. Merker: A Middleware-Based Approach to Database Caching, in: Proc. ADBIS (Thessaloniki), LNCS 4152, Springer, 2006, pp. 184-199.

[5] T. Härder, A. Bühmann: Datenbank-Caching: Eine systematische Analyse möglicher Verfahren, Informatik – Forschung und Entwicklung, Springer (2004)

[6] T. Härder, A. Bühmann: Value Complete, Column Complete, Predicate Complete - „Magic Words Driving the Design of Cache Groups, VLDB Journal, Januar 2007 (DOI 10.1007/s00778-006-0035-9).

[7] J. Klein: Entwicklung einer automatisierten Messumgebung für das Constraint-basierte Datenbank-Caching, Diplomarb., TU Kaiserslautern, Oktober 2006, URL: http://wwwdvs.informatik.uni-kl.de/pubs/DAsPAs/Kle06.DA.html

[8] P.-Å. Larson, J. Goldstein, J. Zhou: MTCache: Mid-Tier Database Caching in SQL Server. ICDE Conference 2004

[9] TPC: TPC-Benchmark W (web commerce) Spezifikation. http://www.tpc.org/tpcw/spec/tpcw V1.8.pdf (2002) Version 1.8.