Vergleich von EJB Applikationsservern

Prof. Dr. Alexander Schill

Dipl.-Inform. Olaf Neumann

Dipl.-Inform. Thomas Springer

Thomas Müller

Abstract

Diese Studie faßt die Ergebnisse der Untersuchung der Applikationsserver Weblogic 5.1.0 (SP5), Inprise Applikationsserver 4.1, Websphere 3.5 (Fixpack 1) und Iona iPortal Application Server 1.2 zusammen. Die Studie ist eine Fortsetzung der Arbeit zum Vergleich von OTMs, welche in der iX 8/2000 veröffentlicht wurde. Im Vordergrund dieser Studie standen Performancebetrachtungen von Applikationsservern anhand eines komplexen Versicherungsbeispiels. In dieser Studie werden neben den Performancediagrammen auch die Erfahrungen beim Umgang mit den Applikationsservern vorgestellt. Weblogic 5.1.0 (SP5) ist in dieser Studie der leistungsfähigste Server. Kurz danach folgt Inprise Application Server 4.1 und mit größerem Abstand Websphere 3.5 (Fixpack 1).

Applikationsserver

Zu Beginn der Studie werden die untersuchten Applikationsserver kurz vorgestellt. Dabei sollen vor allem entscheidende Merkmale verglichen und Besonderheiten der einzelnen Server näher betrachtet werden. Als Kriterien des Vergleiches dienen nicht nur rein technische Aspekte, diese können in den Datenblättern der Hersteller nachgeschlagen werden, es werden vorrangig die Handhabung und das Verhalten der Server im Praxiseinsatz untersucht.

Konkret wird dabei die Installation sowie die Art und Weise der Konfiguration betrachtet. Dazu wird auch die Qualität der mitgelieferten Dokumentation beurteilt.

Aus technischen Gesichtspunkten wird die Fähigkeit der Server im Einsatz als Cluster beschrieben, wobei auch auf die unterschiedlichen Realisierungen von Load Balancing und Failover Mechanismen eingegangen werden soll.

Als wichtiger Aspekt der Handhabbarkeit eines Applikationsservers wird auch die Art und Weise des Deployments beschrieben und kritisch betrachtet.

Weblogic 5.1.0SP5

Die Installation und Konfiguration des Servers ist relativ einfach vorzunehmen und anhand der Dokumentation gut nachvollziehbar. Unter NT steht dafür eine Installationsroutine zur Verfügung. Unter anderen Betriebssystemen kann die Installation auch von Hand erfolgen, so dass der Server z.B. auch unter Linux einsetzbar ist. Die Dokumentation dazu ist ausschließlich auf den Web-Seiten von BEA zu finden. Die Installation des Servicepacks ist dagegen nur unzureichend dokumentiert. Außerdem müssen die mitgelieferten Dateien und die Umgebung manuell installiert und angepaßt werden. Die Installation des Servicepacks ist jedoch für das korrekte Arbeiten des Weblogic Servers unbedingt erforderlich.

Nach erfolgter Installation sollte eine Grundkonfiguration über die Datei *weblogic.properties* vorgenommen werden. Die hier hinterlegten Eigenschaften sind in der Konsole, einer graphischen Oberfläche des Servers, einsehbar. Außerdem werden in dieser Datei auch Benutzer und Rollen definiert. Da diese aber höchstens für einfache Konfigurationen ausreichen, besitzt der Server auch ein sogenanntes *realm* Interface, um die Authentisierung mit beliebigen Systemen durchführen zu können.

Die Konsole dient beim Weblogic Server also weitestgehend zur Überwachung des Servers bzw. des Clusters zur Laufzeit. Konfigurationsänderungen werden in der Datei *weblogic.properties* vorgenommen. Überwacht werden kann dabei sowohl die aktuelle Konfiguration, als auch die momentane Auslastung des Servers. Die Konsole muss nicht zwingend auf derselben Maschine ausgeführt werden, auf der der Weblogic Server läuft, was zur Ressourcenschonung und bei der zentralen Überwachung aller eingesetzter Server im Netz hilfreich ist.

Der Weblogic Server 5.1.0 (SP5) setzt alle Standards der J2EE-Spezifikation um. Als einziger Server bietet er schon Teile der Funktionalität der EJB Version 2.0 an. Bei der Behandlung von teilweise nicht standardkonformen EJB-Implementierungen, zeigt sich der Weblogic Server als sehr tolerant.

Das Clustering von EJBs auf dem Weblogic Server erfolgt automatisch über replica-aware Stubs und ist einfach zu konfigurieren. Dazu müssen mehrere Server clusterfähig gestartet werden, auf denen dieselben Beans unter demselben JNDI-Namen deployed sind müssen. Beim Deployment ist darauf zu achten, dass die Clusterfähigkeit durch Setzen einer Eigenschaft der Beans ermöglicht wird. Zum Load Balancing können verschiedene Verfahren *round-robin* (Standard), *weight-based round robin*, *random* sowie *parameter-based* eingesetzt werden. Zum parameterbasierten Clustering ist die Erstellung eines Callrouters notwendig. Dieser kann Aufrufe anhand übergebener Parameter mittels eines frei zu implementierenden Verfahrens an entsprechende Serverinstanzen weiterleiten.

Eine von BEA empfohlene Clusterstruktur ist die Trennung der Webebene von der Ebene der Applikationsserver. Dazu kann entweder der Weblogic Web-Server unter Einsatz des *HttpClusterServlets* oder ein anderer Webserver in Verbindung mit einem Proxy Plugin verwendet werden. Mögliche Webserver sind dabei Apache, Netscape Enterprise Server und Microsoft Internet Information Server. Mit Hilfe dieser Konfiguration kann ein Clustering der Servlets und JSPs mit Hilfe von replica-

aware Proxies erfolgen. Dies bedeutet, die Webserver verwalten eine Liste von Weblogic Server Instanzen, welche die cluster-fähigen Servlets enthalten und leiten die Aufrufe an diese weiter. Die Verteilung erfolgt dabei über Round Robin.

Eine Beschreibung aller möglichen Formen, um einen Cluster unter BEA Weblogic zu ermöglichen, würde den Rahmen dieser Studie sprengen. Eine gute Übersicht und Konfigurationshilfe findet sich dazu auf der BEA Webseite.

Das Deployment kann einerseits über ein grafisches Werkzeug, andererseits über die Kommandozeile erfolgen. Das grafische Werkzeug ist dabei für die erstmalige Konfiguration gedacht. Hier können die *Deployment Descriptoren* auf einfache Weise erstellt werden. Allerdings kam es zu gelegentlichen Abstürzen des Tools, welches eine Routinearbeit erschwerte. Hierfür ist das Deployment über Kommandozeile eine sehr gute Hilfe. Sind die Descriptoren einmal erstellt, lässt sich über das kommandozeilenbasierte Deployment erheblich Zeit sparen. Um die Beans beim nächsten Serverstart automatisch zu deployen, müssen entsprechende Einträge in der Datei *weblogic.properties* vorgenommen werden. Es ist zwar prinzipiell möglich, ein Hot-Deployment per Konsole vorzunehmen oder bereits deployte Beans zur Laufzeit neu zu deployen, diese Änderungen sind aber nur zur Laufzeit des Servers gültig. Es ist also auf jeden Fall notwendig, die vorgenommenen Änderungen auch in der Datei *weblogic.properties* festzuschreiben. Das grafische Deploymentwerkzeug soll in der Version 6 komplett verändert werden.

Der Server zeichnete sich durch sehr gute Performance und hohe Stabilität in den Tests aus. Die Stabilitätsgarantie kann zudem durch Failover und Clustering erhöht werden. Dabei wird Failover, bis auf den Status bei *stateful SessionBeans*, deren Unterstützung in der Version 6 angekündigt ist, vollständig unterstützt.

Der Ressourcenbedarf des Weblogic Servers ist relativ gering. Der Server ist vollständig in Java implementiert. Für einige Betriebssysteme werden außerdem Performance Packs angeboten, die als Maschinencode für das jeweilige System vorliegen. Der Server inclusive der Testanwendung verbraucht ca. 70 Mbyte Hauptspeicher. Darin sind sowohl die EJB, als auch die weiteren, von Weblogic angebotenen, Dienste enthalten.

Für den Weblogic Server existieren eine Reihe von Zusatztools, sei es zum Anschluß objektorientierter Datenbanken (z.B. von Versant), zum OR-Mapping (von Toplink) und zum Caching (z.B. Javlin) etc. BEA bietet passend zum Server die Entwicklungsumgebung WebGain (ehemals Visual Cafe) an.

Inprise Application Server 4.1

Wie schon bei Weblogic erwies sich die Installation auch beim Inprise Application Server als sehr einfach. Die Onlinedokumentation bietet Hilfe für Standardinstallations- und Konfigurationsschritte, Detailinformationen für eine tiefgründige Problemanalyse sind jedoch in der Dokumentation kaum enthalten. Hierzu kann auf die Newsgroup zum Inprise Application Server verwiesen werden, welche für die meisten aufgetretenen Probleme eine gute Hilfestellung bot.

Nach der Installation können alle Einstellungen über die Konsole vorgenommen werden. Eine Bearbeitung der durchaus vorhandenen Konfigurationsdateien erwies sich nicht als zwingend notwendig. Nur bei der Einbindung von Datenbanktreibern ist

neben dem Einspielen der Files auch die Bearbeitung von Konfigurationsdateien notwendig. Dabei sind die Änderungen in Abhängigkeit von der Konfiguration in mehreren Files vorzunehmen, z.B. für den Webserver und den Container.

Inprise erlaubt die komplette Konfiguration über die Konsole. Dabei werden die Änderungen in den entsprechenden Files gespeichert.

Der Inprise Application Server unterstützt den J2EE-Standard ebenfalls vollständig. Insbesondere verlangt der Server eine streng standardkonforme Implementierung der Beans. Darüber hinaus basiert der Server komplett auf CORBA und der Visibroker Technologie. Der Visibroker bietet zusätzlich einen POA und Object-by-Value Technologien. Als Namensdienst wird der CosNaming-Dienst eingesetzt.

Der Inprise Application Server unterstützt das Rollenkonzept des EJB Standards. Um flexibel Sicherheitsanforderungen umzusetzen, kann genau wie beim Weblogic Server, ein sogenanntes *realm* Interface eingesetzt werden. Verlangt die Anwendung zusätzliche Anforderungen an die Sicherheit, bietet Inprise die Möglichkeit, eine externe Benutzerverwaltung mittels der CORBA Schnittstelle anzubinden.

Das Clustering erfolgt beim Inprise Application Server über den JNDI-Dienst. Dabei wird nur eine Instanz des Namensdienstes im zu errichtenden Cluster gestartet. Dieser verteilt dann die Clientanfragen nach der Round-Robin Methode auf die beteiligten Serverinstanzen. Voraussetzung dafür ist, dass die Beans auf den einzelnen Servern unter demselben JNDI-Namen ansprechbar sind und die Eigenschaft vbroker.naming.propBindOn auf 1 gesetzt wurde. Inprise unterstützt das Clustering von Stateless Session Beans und laut eigenen Angaben das Clustering von Stateful Session Beans über einen zusätzlichen Session-State Service. Das konnte im vorliegenden Testfall aber nicht überprüft werden, da hier keine Stateful Session Beans Verwendung fanden.

Um Ausfälle des Namensdienstes behandeln zu können, bietet Inprise die Möglichkeit zwei Namensdienste nach dem Master-Slave Modell zu starten. Beide Namensdienste müssen mit den gleichen Registrierungsdaten arbeiten, Caching darf von beiden Namensdiensten nicht verwendet werden. Bei einem Ausfall des Master-Dienstes übernimmt der Slave dessen Funktion. Dies erfolgt für den Client transparent.

Das Deployment erfolgt ebenfalls mit Hilfe der grafischen Konsole. Dabei kann entschieden werden, ob die Einhaltung des J2EE Standards streng oder weniger streng überprüft wird. Nach erfolgtem Deployment kann ein Client-Jar erstellt werden, in dem die benötigten Stubs enthalten sind. Wurde ein Bean deployed, so ist es auch nach dem Neustart des Servers vorhanden und kann nur durch explizites Entfernen mit Hilfe der Konsole aus dem Server gelöst werden.

Der Hauptspeicherverbrauch des Inprise Application Servers liegt mit allen Beans und Diensten nach dem Serverstart bei 133 Mbyte. Durch ein vermutetes Memory Leak steigt der Verbrauch an Hauptspeicher im laufenden Betrieb aber stark an und kann zu OutOfMemory Fehlermeldungen führen.

Der Inprise Application Server bietet darüber hinaus weitgehende Möglichkeiten den Container auszutauschen bzw. Einfluss darauf zu nehmen. Dafür wird eine eigene Schnittstelle bereitgestellt.

Die von Inprise angebotene Entwicklungsumgebung JBuilder Enterprise erlaubt das direkte Deployment sowohl in den Inprise Application Server aber auch in den Weblogic Server. Sie bietet zahlreiche Wizards zum einfachen Erstellen von Servlets, JSPs inklusive Fehlerbehandlung, Beans, Testanwendungen und dem Zugriff auf Datenbanken, welche das Entwickeln von Anwendungen vereinfachen.

Probleme gab es bei der Testdurchführung unter Inprise Application Server 4.1.0. Dieser brach, nachdem der Test eine Weile lief, mit der oben beschriebenen OutOfMemory Fehlermeldung ab. Recherche in Newsgroups ergab, dass es sich dabei um ein bekanntes Memory-Leak handelt. Bei nachfolgender Verwendung des Inprise Application Servers 4.1.1 trat der Fehler in unserem Testumfeld nicht mehr auf, obwohl in Newsgroups immer noch davon berichtet wird.

IBM Websphere 3.5 Fixpack 1

Die Serverinstallation gestaltete sich in dem hier vorhandenen Umfeld recht einfach, einige Schwierigkeiten traten dagegen beim Einspielen des Fixpacks auf. Dieser Vorgang ist skriptbasiert und birgt einige Fehlerquellen, deren Ursachen nur ungenügend dokumentiert wurden. So war z.B. auf den verwendeten NT-Servern Deutsch als Standardsprache eingestellt. Das führte dazu, dass das Skript den vorhandenen freien Festplattenspeicher falsch berechnete und somit die Bearbeitung abbrach. Ein weiterer Fehler dabei war z.B., das sich das zur Installation des Fixpacks verwendete JDK selbst aktualisieren wollte, was zu Sharing Violations führte.

IBM Websphere verwendet zur Speicherung von Konfigurationsdaten und Laufzeitinformationen eine administrative Datenbank. Dies ist standardmäßig die mitgelieferte DB2-Datenbank. Bei Verwendung dieser Konfiguration ergaben sich jedoch gravierende Performanceprobleme.

Um die Performance des IBM Websphere zu erhöhen, wurde Oracle als administrative Datenbank eingesetzt. Ein entsprechendes Werkzeug zum Ändern der administrativen Datenbank kann von der IBM-Webseite heruntergeladen werden. Bevor die Datenbank umgestellt werden kann, muss man allerdings den Inhalt der derzeit verwendeten Datenbank mit Hilfe des mitgelieferten Tool XMLConfig oder aber unter Verwendung der Konsole als XML-Datei sichern. Auch bei diesem Prozess ist die mitgelieferte Dokumentation nicht nur lückenhaft, sondern sogar falsch. Dort heißt es, man solle vor dem Export der Konfiguration den administrativen Server stoppen. Richtig wäre es, den Server zu starten, da sonst kein Export vorgenommen werden kann. Mit Hilfe des Werkzeuges DBUpgrade kann dann der Datenbanktreiber auf oracle.jdbc.driver.OracleDriver umgestellt werden. Danach muss man in admin.config noch das Initialisieren der neuen und derzeit noch leeren Datenbank aktivieren und kann dann den in XML sichergestellten alten Datenbankinhalt wieder einspielen. Auch dazu muss der administrative Dienst gestartet werden. Die Änderung der administrativen Datenbank bzw. deren Leistung haben entscheidenden Einfluss auf die Geschwindigkeit der Ausführung des Servers, da dort wichtige Statusinformationen ständig aktuell gehalten werden.

Die Dokumentation des Websphere Servers ist zum Lösen von Problemen völlig unzureichend. Nur ungenügend Hilfestellung erhält man bei Konfigurationsschritten, die über das standardmäßige Installieren und Konfigurieren hinausgehen. Es fehlt an vielen Details und Hintergrundinformationen. Ein Beispiel dazu ist das bereits weiter

oben beschriebene Ändern der Datenbank oder das Einspielen des Fixpacks. Weitere Lücken ergeben sich beim Einrichten eines Clusters, wobei sich aus der Dokumentation zwar die ungefähre Vorstellung der Funktionsweise des Websphere Clusters ergibt, nicht jedoch die konkreten Konfigurationsschritte, um diesen auch wirklich zu installieren. Ausgesprochen hilfreich ist aus diesen Gründen die Websphere Newsgroup.

Der Server hat einen hohen Ressourcenbedarf, der durch den Einsatz einer Datenbank (Administrative Datenbank) zur Speicherung der Konfiguration und von Laufzeitdaten noch erhöht wird. So verbraucht allein der administrative Dienst nach seinem Start ca. 50 MB Hauptspeicher. Die zum Betrieb notwendige Konsole verbraucht ca. 113 MB und nach dem Start einer Instanz des Applikationsservers werden zusätzliche 34 MB in Anspruch genommen. Um einen fairen Vergleich zu bekommen muss man noch den Verbrauch des zusätzlich zu startenden Web-Servers einrechnen, welcher in der Grundkonfiguration ca. 10 MB verbraucht.

Wie die anderen Server besitzt Websphere eine Konsole zur Konfiguration und zum Deployment, welches allerdings sehr langsam ist, so dass eine flüssige Arbeit erschwert wird.

In der Konsole können sämtliche, den Applikationsserver betreffenden, Einstellungen vorgenommen werden. Die Darstellung erfolgt baumartig, wobei als Wurzel die Administrationsdomäne dient. Unter dieser sind alle in der Domäne vorhandenen Serverinstanzen aufgeführt, welche sich individuell bzw. als Cluster konfigurieren lassen. Des weiteren lassen sich die benötigten Datenbanktreiber und Datasources konfigurieren, die dann den entsprechenden Beans zugewiesen werden können.

Sollen für die Anwendung spezielle Sicherheitseinstellungen vorgenommen werden, lassen sich diese ebenfalls in der Konsole einstellen. Alle vorgenommenen Änderungen sind auch über einen Neustart des Servers verfügbar, müssen also nicht zusätzlich festgeschrieben werden.

IBMs Websphere Advanced Edition Applikationsserver unterstützt nicht komplett die J2EE-Spezifikation. Dies wird vor allem bei der Umsetzung des EJB 1.1 Standards deutlich, der z.Zt. immer noch nicht vollständig umgesetzt wird. Obwohl EJB1.1 nicht vollständig unterstützt wird, waren dennoch keine weiteren Kodeanpassungen notwendig, da weder Collections noch Sicherheitsspezifische Operationen von Java2 verwendet wurden. Ein Message-Dienst steht nur in der Enterprise Version zur Verfügung.

Websphere unterstützt verschiedene Sicherheitsmechanismen, die mit Hilfe der Konsole implementiert werden können. Dabei werden Security Policies definiert, die mit Hilfe des Security Services überprüft werden. Die Security Policies können dabei auf einer Security-Implementierung durch das Betriebssystem, auf LDAP oder auf eine proprietäre Implementierung beruhen.

Das Clustering wird vom Server mit Hilfe der administrativen Datenbank unterstützt. Hierbei müssen alle am Cluster beteiligten Administrationsserver dieselbe administrative Datenbank benutzen. Ausgehend von einem konfigurierten Applikationsserver können von diesem Clones erstellt werden. Diese können dann auf derselben physischen Maschine oder auf anderen Maschinen installiert werden. Zusätzlich zu den Serverclones müssen die EJB's WLM-enabled werden, d.h. es

werden auf dem Client sogenannte Smart-Stubs installiert, welche die verschiedenen Server ansprechen und somit das Load Balancing bzw. Failover ermöglichen. Zur Anfrageverteilung stehen die Methoden Round Robin und Random zu Verfügung, die zusätzlich mit der Option, den lokalen Clone zu bevorzugen, versehen werden können. IBM unterstützt derzeit das Clustering von Stateless Session Beans und Servlets bzw. JSPs. Stateful Session Beans sind immer einem bestimmten Container zugeordnet.

Das Deployment gestaltet sich bei Websphere als sehr träge, sowohl bei der Durchführung der Schritte vor dem Deploymentprozeß, als auch beim eigentlichen Deployment. Es wird die Möglichkeit angeboten, mehrere Beans in einem JAR-Archiv gleichzeitig zu deployen. Diese Vereinfachung (hat sich in unserem Testumfeld jedoch als nicht funktionsfähig erwiesen, was zur Folge hatte, dass das Deployment jedes Beans einzeln erfolgen musste. Nach erfolgreichem Deployment sind die Beans auch über Serverneustarts hinaus im Server verankert. Websphere arbeitet darüber hinaus beim Deployment außerdem noch mit den serialisierten Deployment-Descriptoren der EJB-Version 1.0, was ein Hauptziel des J2EE Standards, die Wiederverwendung der Komponenten, stark beeinträchtigt, da die Descriptoren neu erzeugt werden müssen. Dafür wurde das interne Deploymenttool verwendet, welches allerdings instabil arbeitet und nicht nachvollziehbare Fehlermeldungen hervorbringt. Zudem erfordert das Ändern der Konfiguration ständig den Neustart des Containers bzw. der Web-Engine, was wiederum zeitaufwendig ist.

Von IBM wird Visual Age als Entwicklungsumgebung angeboten, die ein direktes Deployment der entwickelten Komponenten in den Websphere Applikationsserver ermöglicht. In der Enterprise Edition stehen dem Applikationsserver weitere Tools zur Verfügung, die allerdings in der Studie nicht Gegenstand der Tests waren.

Iona

Die grundlegende Installation des Servers ist unter Windows NT Server recht einfach durchzuführen. Nach der Installation traten aber schon die ersten Probleme auf. So musste auf dem verwendeten Rechner die durch Iona mitgelieferte Datei orb.properties per Hand in das entsprechende JDK Verzeichnis kopiert werden. Die Konfiguration des Servers wird zur Installationszeit abgefragt und liefert einen lauffähigen Server.

Die mitgelieferte Konsole dient zum Erstellen der Anwendungen. Eine Konfiguration des Servers sowie das eigentliche Deployment ist damit nicht möglich.

Insgesamt weist die Arbeit mit der Konsole einige Fehler auf. So wurden bei unserem Beispiel die PKs bereits hinzugefügter Entity Beans gelegentlich beim Hinzufügen neuer Entity Beans mit dem PK des neu hinzugefügten überschrieben. Werden einzelnen Methoden spezielle Transaktionsattribute zugewiesen, so traten hierbei auch merkwürdige Verhaltensweisen auf. Existieren in einem Bean zwei gleiche Methoden mit unterschiedlichen Parametern, so wurden die Parameter bei der Transaktionszuweisung ignoriert und im Deployment Descriptor nur eine der beiden Methoden mit Transaktionsattributen versehen. Dies führte dann beim eigentlichen Deployment zwangsläufig zu Fehlermeldungen. Abhilfe konnten wir nur durch das Setzen des Attributes TransactionRequired auf alle Bean Methoden schaffen.

Schwierig zu behandeln war auch das Erzeugen von Web-Applikationen. Dazu mussten alle benötigten Servlets und JSPs in die Konfiguration eingetragen werden. Das erste Problem ergab sich dabei an dem geforderten Vorkompilieren der Beans. IONA Applikationsserver ist der einzige Server im Testfeld, der ein Vorkompilieren der JSPs zwingend erfordert. Der dazu benötigte Schalter am Deploymenttool ist nicht dokumentiert.

Des weiteren müssen alle, durch die Servlets benötigten Klassen, in der erstellten *.war Datei eingetragen sein. Obwohl die Klassen im Laufe der Erstellung der Anwendung in der Konsole angegeben wurden, sind diese nicht zum Web-Archiv hinzugefügt worden. Diese müssten dann per Hand zum Archiv hinzugefügt werden, was sehr umständlich war.

Das Deployment bei Iona ist insgesamt nur unzureichend dokumentiert. Mit Hilfe der Konsole ist, wie bereits angesprochen, die Anwendung zu erstellen. Dabei kann es sich entweder um eine reine EJB-Anwendung handeln, welche dann in Form eines JAR-Archives vorhanden ist, oder aber um eine Web-Anwendung, welche zusätzlich die JSPs und Servlets enthält und als WAR-Archiv vorliegt.

Auf den Server deployed wird die fertige Anwendung mit Hilfe eines Kommandozeilentools. Iona unterstützt kein Hot-Deployment. Die Beans sind nur für die Laufzeit des Servers verfügbar, nach einem Neustart muss der Deployment-Prozess wiederholt werden.

Iona konnte leider nicht mehr in die Tests eingebunden werden, da es unlösbare Probleme bei der Anbindung der DB2 Datenbank gab. Iona verwendet zur Datenbankanbindung die mitgelieferten Treiber der Firma Merant. Diese sind derzeit nur für Oracle verfügbar. Eine neue Version, welche unter anderem auch DB2 unterstützen soll, ist zwar von Merant angekündigt, befindet sich aber zur Zeit noch im nicht öffentlichen Beta Stadium, so dass dieser nicht verwendet werden konnte.

Da die im Test verwendeten Beans alle mit Bean Managed Persistence arbeiten, könnte die Datenbankanbindung auch direkt in den Beans vorgenommen und somit der Container als Datenbankvermittler umgangen werden. Diese Konfiguration setzt aber den Verzicht auf Connection Pooling durch den Container voraus, welches die Performance negativ beeinflusst hätte und die Vergleichbarkeit der Ergebnisse in Frage gestellt hätte.

Der Iona Application Server setzt den J2EE Standard vollständig um und ist dabei streng standardkonform. Abweichungen der Beans vom EJB Standard werden kaum toleriert.. Allerdings setzt Iona in Bezug auf Sicherheit nur das im EJB Standard beschriebene Rollenkonzept um und bietet somit nur einen ungenügenden Sicherheitsmechanismus, da damit keine anderen Sicherheitssysteme einbezogen werden können.

Iona bietet auch in der neuesten Version keine direkte Clustering-Unterstützung, was den Einsatz in hochverfügbaren Anwendungen ausschließt. Es existiert lediglich die Möglichkeit, durch die Konfiguration des Orbix2000 ein Cluster zu erhalten, was in dieser Studie aber nicht betrachtet wurde, da Iona aus den bereits genannten Gründen, in unserem Testumfeld, nicht einmal als Single Server betrieben werden konnte.

Iona benötigt zum Betrieb des eigentlichen Servers Dienste, die vor dem Start des Servers gestartet werden müssen. Dazu zählt zum einen der Locator-Dienst, der einen Speicherverbrauch von ca. 17 Mbyte hat und der Naming Dienst, dessen Speicherverbrauch ca. 14 Mbyte beträgt. Der Application Server selbst verbraucht nach dem Start ca. 14 Mbyte. Insgesamt ist der Iona Application Server damit der Applikationsserver mit dem geringsten Ressourcenverbrauch.

Die Dokumentation liegt bei Iona zwar in gedruckter Form vor, ist aber völlig unzureichend. Wie schon beim Inprise Application Server und IBM Websphere lässt die Dokumentation auch hier viele Fragen hinsichtlich spezieller Konfigurations- und Handhabungsschritte offen.

Iona bietet selbst keine Entwicklungsumgebung an. Die Beans müssen also mit einer anderen Entwicklungsumgebung erzeugt und mit Hilfe der Konsole zu einer Anwendung zusammengestellt und anschließend, wie bereits beschrieben, mit Hilfe des Deploymenttools bearbeitet werden.

Testumgebung

Als Testumgebung diente das NT-Labor des Lehrstuhles Rechnernetze der TU Dresden. Dieses besteht aus mehreren Dual-Pentium Rechnern, die über ein 100Mbit LAN (switched) verbunden sind. Als Betriebssystem wurde NT4.0 (SP6a) verwendet. Alle Applikationsserver verwendeten als Datenbank eine IBM Universal Database 7.0, welche auf einem Windows NT 4.0 Server (SP6a) System installiert wurde. Diese Datenbank wurde für die Nutzdaten verwendet. Als Treiber wurde der jdbc.db2.net.Driver verwendet. Dieser Treiber ist ein XA-fähiger JDBC2.0-Treiber.

Es wurde darauf geachtet, dass für die Server möglichst gleiche Bedingungen vorliegen. Dies betrifft zum einen die Nutzung der gleichen Testdaten sowie die Nutzung derselben Konfiguration. Damit im Test überall die gleichen Treiber genutzt werden, wurde der Treiber explizit in allen Applikationsservern auf den DB2-Treiber db2.jdbc.net.Driver gesetzt und die Größen der Pools angepasst.

Um den Namensdienst zu finden, waren bei den Clients außerdem Anpassungen zum Finden des InitialContext notwendig, da z.T. verschiedene Protokolle eingesetzt werden.

Die Applikationsserver liefen ebenfalls unter Windows NT4 Server (SP6a) aber auf anderen Maschinen als die Datenbank. Als Java Laufzeitumgebung wurde SUN JDK1.2.2 verwendet. Obwohl auch einige der getesteten Server das neuere JDK 1.3 unterstützen, wurde aus Vergleichsgründen auf allen Servern die gleiche JDK Version benutzt.

Die Hardwareausstattung der am Versuch beteiligten Rechner ist in der nachfolgenden Tabelle aufgeführt. Alle Applikationsserver liefen für die jeweils gleichen Tests auf den gleichen Maschinen. Alle Rechner besitzen ein SCSI160 Interface.

	Prozessor	**RAM**
DB Server	2x Intel Pentium III 667 MHz	256 MB
Applikationsserver 1,2	2x Intel Pentium III 800 MHz	512 MB
Applikationsserver 3	2x Intel Pentium III 667 MHz	256 MB
Client Rechner	2x Intel Pentium III 667 MHz	256 MB

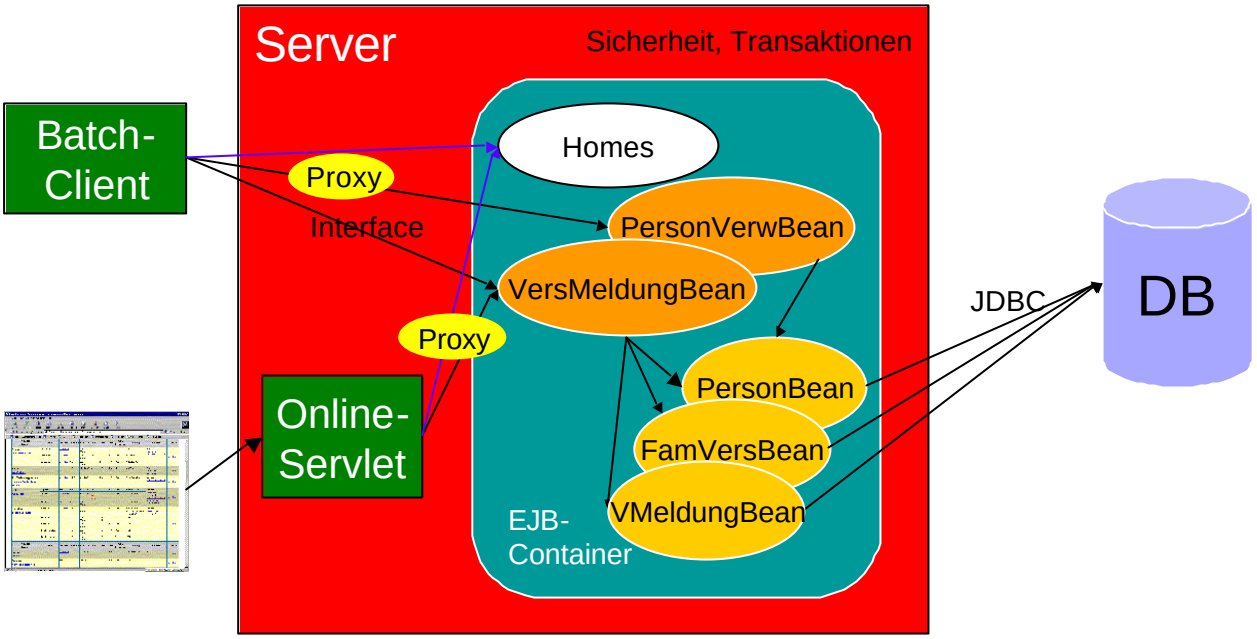
Testbeispiel

Im folgenden wird das Testbeispiel näher vorgestellt. Das Beispiel stammt aus dem Versicherungsbereich und bearbeitet Vorgänge zum Anmelden/Abmelden von Arbeitnehmern und zugehörige Operationen. Es besteht aus 4 Session- und 3 EntityBeans, welche die Business-Logik repräsentieren und 14 JSPs und 3 weiteren Servlets zur Anzeige. Darüber hinaus werden Proxy-Objekte eingesetzt, um die Performance zu verbessern. Die Proxy-Objekte werden von den EntityBeans erzeugt und liefern alle Daten des EnityBeans in einem Objekt. So können die gesamten Daten eines Entity Beans bei Methodenaufrufen in einem Objekt übergeben werden., z.B. an ein SessionBean. Dieses erspart mehrfache, zeitaufwendige Zugriffe, welche besonders dann auftreten, wenn von einem Versicherten alle Daten auf dem Bildschirm visualisiert werden sollen.

Dieser Anwendungsfall ist in dem verwendeten Beispiel auch der überwiegend auftretende. Es handelt sich also in der Mehrzahl um Lesevorgänge, wobei aber auch Schreibvorgänge berücksichtigt werden müssen.

Es existieren zwei Clients für die Nachbildung des Online- und Batchbetriebs der Versicherung, ein sogenannter Batch-Client, der eine größere Menge von Anfragen hintereinander direkt an die Beans stellt und ein Online-Client, der Anfragen an Servlets und JSPs stellt und somit mehrere Browser simuliert. Der Batch-Client stellt den Anwendungsfall dar, in dem ein bestimmter Betrieb nicht permanent Online ist und die Daten über Versicherungsmeldungen sammeln muss. Beim Batch-Client werden die Versicherungsmeldungen mit einmal Mal in das System eingespielt. Der Online Client steht für den noch häufiger verwendeten Fall, dass die Versicherungsmeldungen in den Betrieben schriftlich erfasst werden und dann durch einen Mitarbeiter der Versicherung Online in das System eingegeben werden. In den Untersuchungen wurden nur Tests mit dem Online-Client vorgenommen. Damit wird ein komplexer Anwendungsfall erreicht der gleichzeitig die EJB und die Web Schicht anspricht.

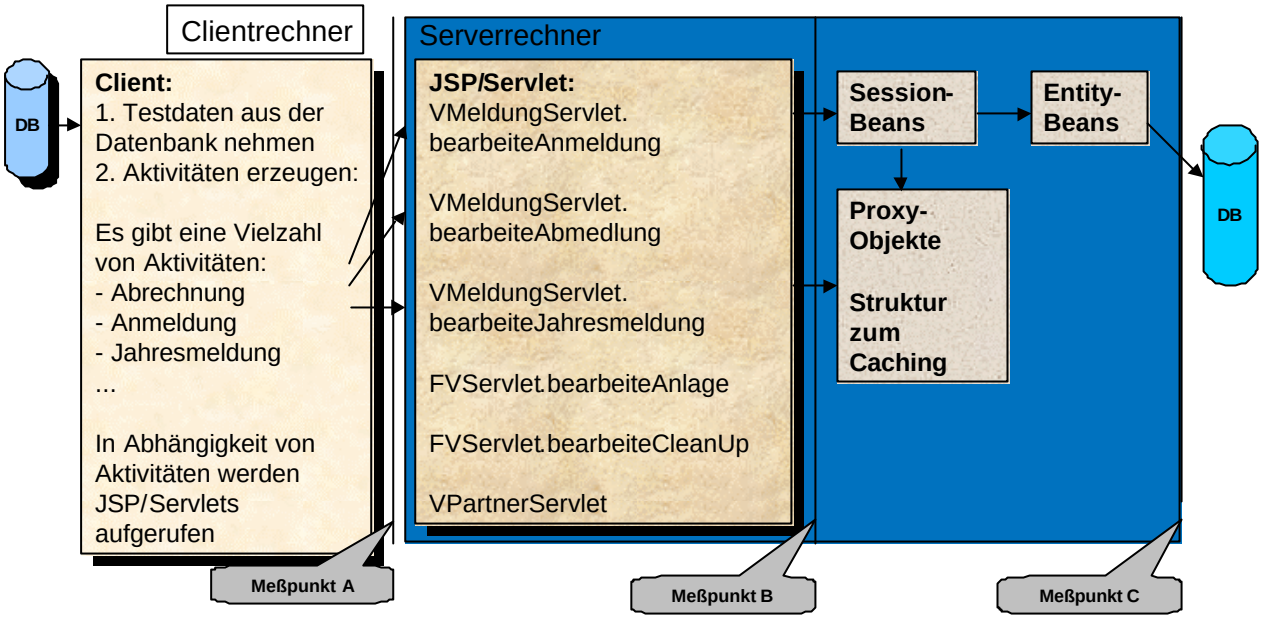
Die Clients simulierten dabei verschiedene Vorgänge, z.B. das Abarbeiten einer Versicherungsmeldung, die An- und Abmeldung Familienversicherter etc. Die Anfragedaten für die Vorgänge werden ebenfalls aus einer anderen Tabelle des generierten Datenbestandes ausgelesen. Dort wurden sie als separate Vorgänge abgelegt.



Um die geforderte Anzahl von Anfragen der Clients bearbeiten zu können, wurden die Server im Cluster betrieben. Die obige Abbildung zeigt, wie z.B. BEA in diesem Fall mit den replica-aware Stubs in einem Cluster arbeitet. Die anderen Applikationsserver verwenden die in den vorangegangenen Abschnitten vorgestellten Techniken des Clustering. Der Cluster arbeitete gegen eine DB2-Datenbank, in die ein generierter Anfangsdatenbestand per Skript importiert wurde.

Der Ablauf einer Versicherungs-Transaktion in dem untersuchten Beispiel wird im folgenden genauer erläutert. Aus dem generierten Datenbestand wurde ein Vorgang z.B. Anmelden/Abmelden eines Versicherten oder Familienversicherten oder Suche eines Versicherten eingelesen. Dieser wurde als eine Versicherungsmeldung, die die entsprechende Operation enthält, an ein Servlet geschickt. Das Servlet selbst ruft wieder Methoden auf den dazugehörigen SessionBeans auf. Dabei beginnt die eigentliche Servertransaktion. Das SessionBean arbeitet nun in Abhängigkeit der vorliegenden Operation verschiedene Funktionen ab. Dies ist z.B. die Speicherung einer neuen Meldung mit dazugehöriger Konsistenzkontrolle. Diese Konsistenzprüfung erfolgt in Abhängigkeit zum jeweiligen Anwendungsfall und kann z.B. das Prüfen der Existenz der angegebenen Personen oder das Heraussuchen einer Anmeldung zu einer Abmeldung sein .

Nach Abschluss der Arbeit des SessionBeans werden die Werte vom Servlet empfangen und zur Darstellung aufgearbeitet. Nach einer einstellbaren Pause beginnt der gesamte Vorgang wieder mit dem Auslesen des nachfolgenden Vorgangs aus dem generierten Datenbestand.



Um das Verhalten der verschiedenen Komponenten in dem System besser beurteilen zu können, wurden verschiedene Messpunkte integriert.

- Messpunkt A

  bestimmt die gesamte Roundtrip-Zeit, die nach dem Auslesen der Operation aus dem generierten Datenbestand für die gesamte Verarbeitung auf dem Server benötigt wird einschließlich dem Empfangen der Anzeigedaten. Dieser Messpunkt befindet sich auf dem Client und misst somit die gesamte Bearbeitungszeit einer Transaktion. Da die Transaktionen stark unterschiedliche Komplexität und somit unterschiedliche Laufzeiten besitzen, wird hier eine Aussage über die Mittelwerte der Messergebnisse getroffen.

- Messpunkt B

  wurde in den Servlets angebracht, um die Zeit der Verarbeitung in den Beans zu bestimmen. Dabei handelt es sich im eigentlichen Sinne nicht um einen, sondern um mehrere Messpunkte, die jeweils an den Aufrufen der Beans im Servlets platziert sind. Verglichen werden können die Punkte anhand eines über alle Messpunkte gebildeten Mittelwerts oder anhand einer direkten Gegenüberstellung der einzelnen konkreten Messpunkte. Um eine konkrete Aussagen zu treffen, ist aber der Vergleich der Mittelwerte ausreichend.

- Messpunkt C

  bestimmt schließlich die Zeit, die für die Bearbeitung innerhalb der Datenbank benötigt wird. Hierbei handelt es sich wiederum um eine Vielzahl von Messpunkten, die über einen Mittelwert aller Messpunkte verglichen wurden. Dieser Wert sollte bei allen Servern in etwa identisch sein, da dieselben Datenbanken und die dazugehörigen Treiber eingesetzt wurden. Geringe Abweichungen können durch unterschiedlich gutes Datenbankpooling der EJB-Server zustande kommen.

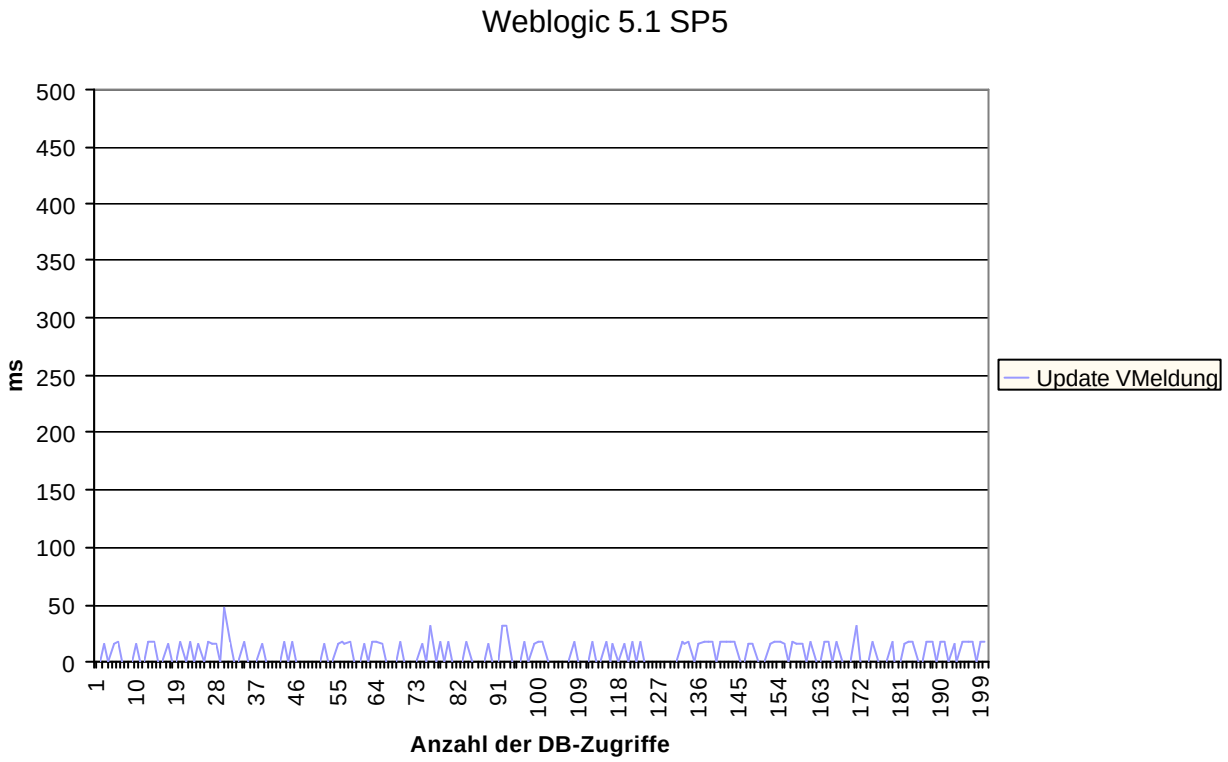
Alle Messdaten wurden in Dateien geschrieben, die zur besseren Auswertung später in Excel übertragen wurden. Zu jedem Messpunkt wurde eine extra Datei aufgebaut. Wie bereits angesprochen gehören zu Meßpunkt B und C mehrere Dateien. Diese wurden zur Auswertung als Mittelwerte zusammengefasst.

Das gesamte Beispiel basiert auf BMP (bean managed persistence), so dass die Zeitmessungen für Meßpunkt C vor und nach den SQL-Anfragen implementiert werden konnten. Der Hauptgrund zur Entscheidung für BMP liegt in der Art der Attributabbildung der Entity Beans auf die Datenbank. Hierbei liegt bei einigen Entity Beans keine 1:1 Zuordnung vor, d.h. ein Bean benötigt für die Speicherung seiner Attribute mehrere Zeilen einer Tabelle. Diese Art der Speicherung wird von den Applikationsservern nicht direkt unterstützt. Somit ist der Einsatz von Container Managed Persistence nicht möglich.

Vergleich

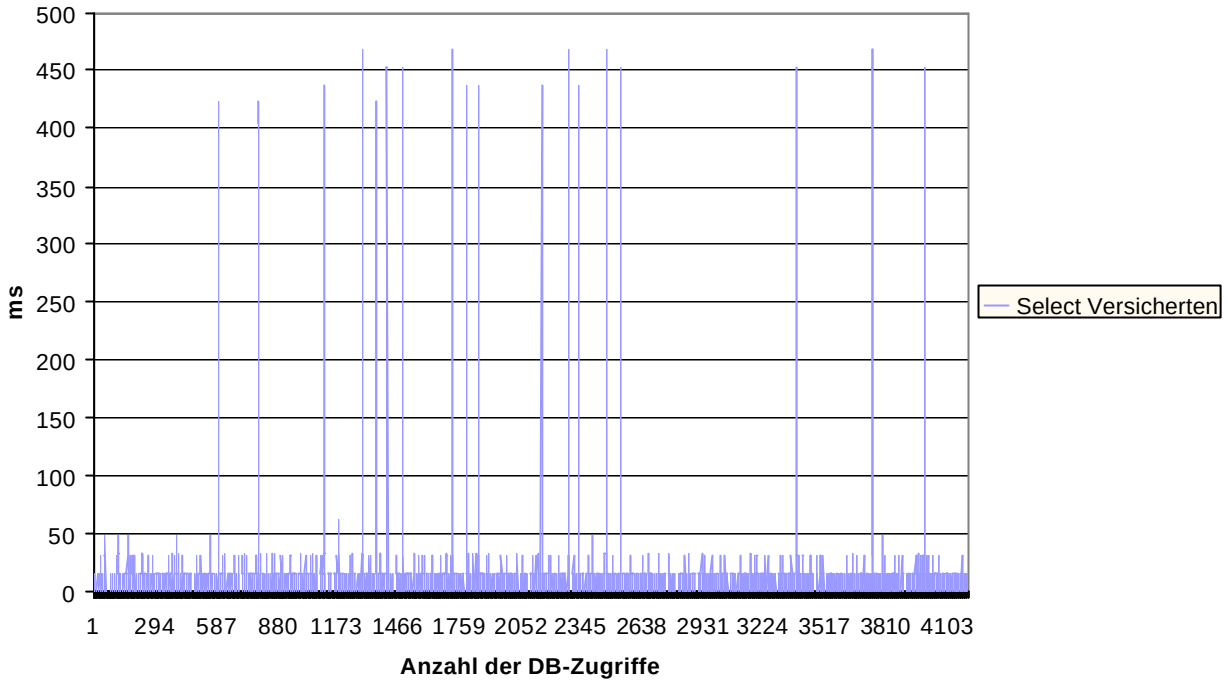
Die folgenden 6 Diagramme geben jeweils für eine Operation auf der Datenbank die Zeiten wieder. Dabei wird zuerst immer ein UPDATE einer Versicherungsmeldung auf der Datenbank betrachtet, danach ein SELECT einer Versicherungsmeldung, wie es z.B. beim Abmelden benötigt wird. Dabei fällt auf, dass beim SELECT einige Spitzen auftreten, während das UPDATE gleichbleibend schnell verläuft. Dies hängt mit dem weitaus häufigeren Auftreten von SELECT Operationen auf der Datenbank

zusammen. Demzufolge können nicht alle benötigten Daten im Cache gehalten werden, sondern müssen zum Teil erst von der Datenbank angefordert werden, was in einem höheren Aufwand für den Zugriff resultiert.

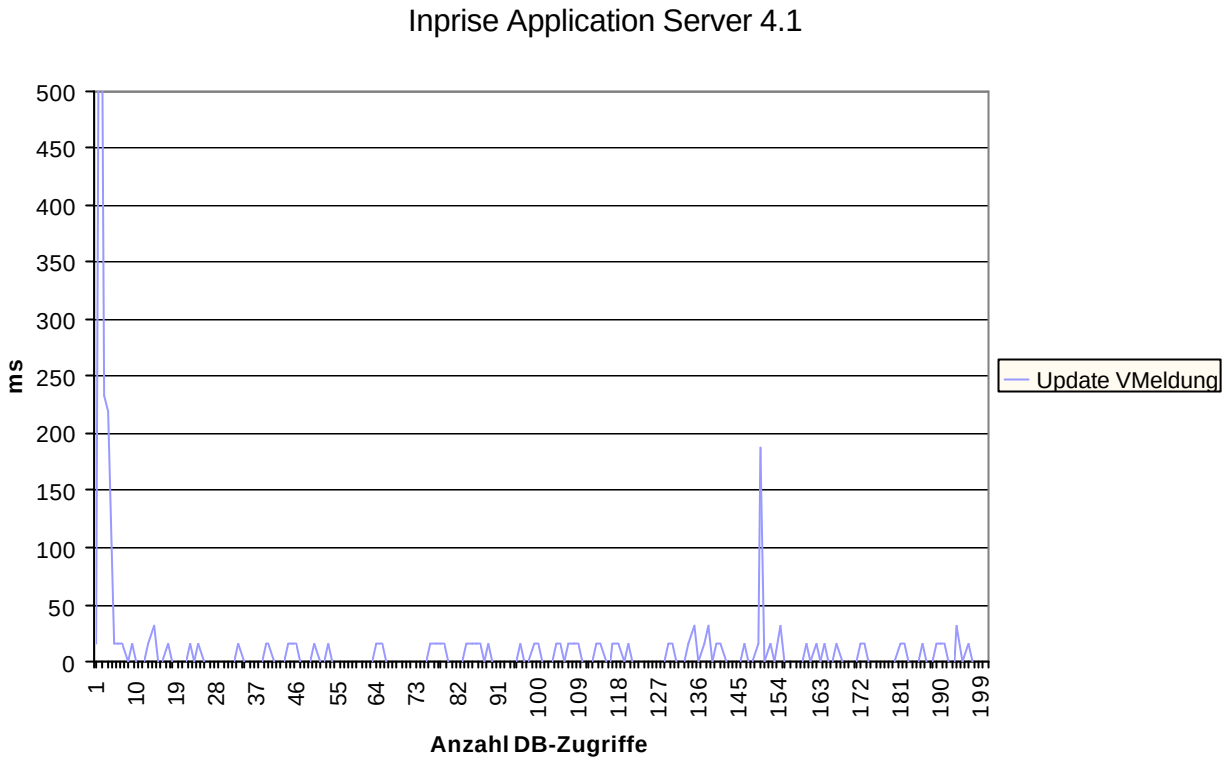


Die SELECT Anweisungen treten über 20 mal so häufig auf. Die Spitzen deuten auf die beschriebenen fehlenden Daten im Cache hin.

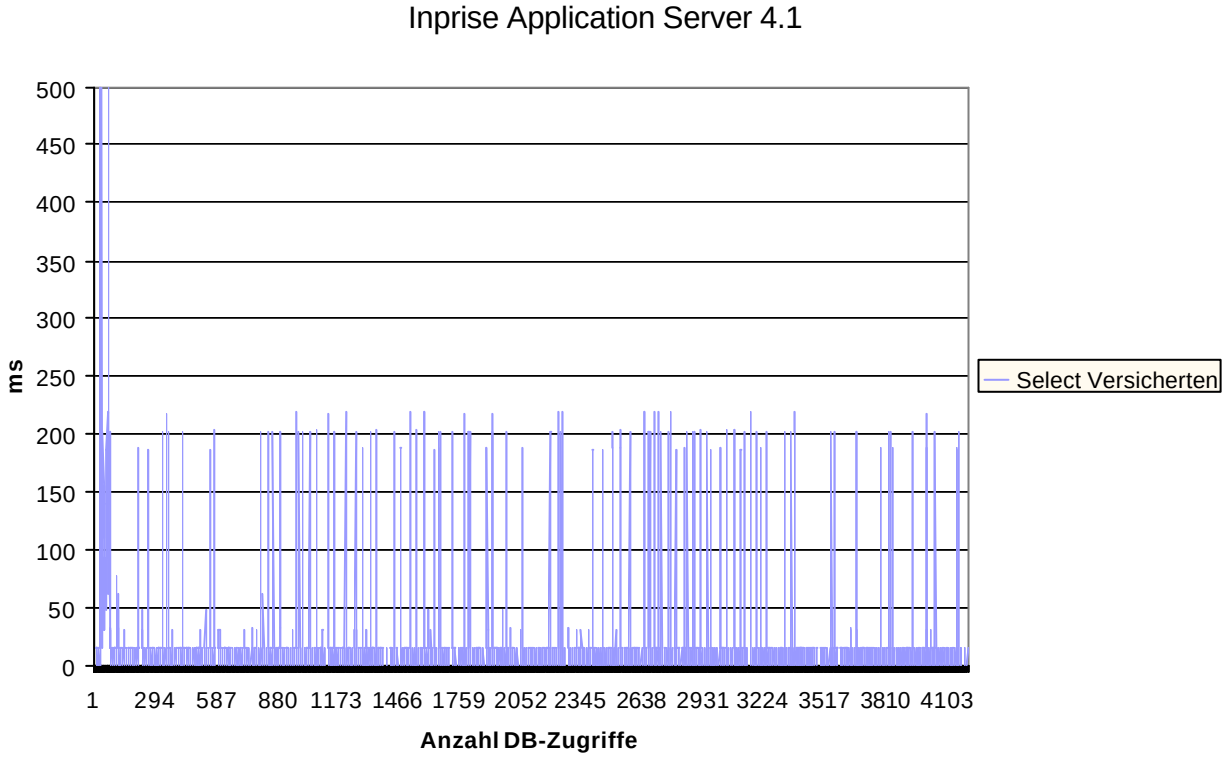
Weblogic 5.1 SP5



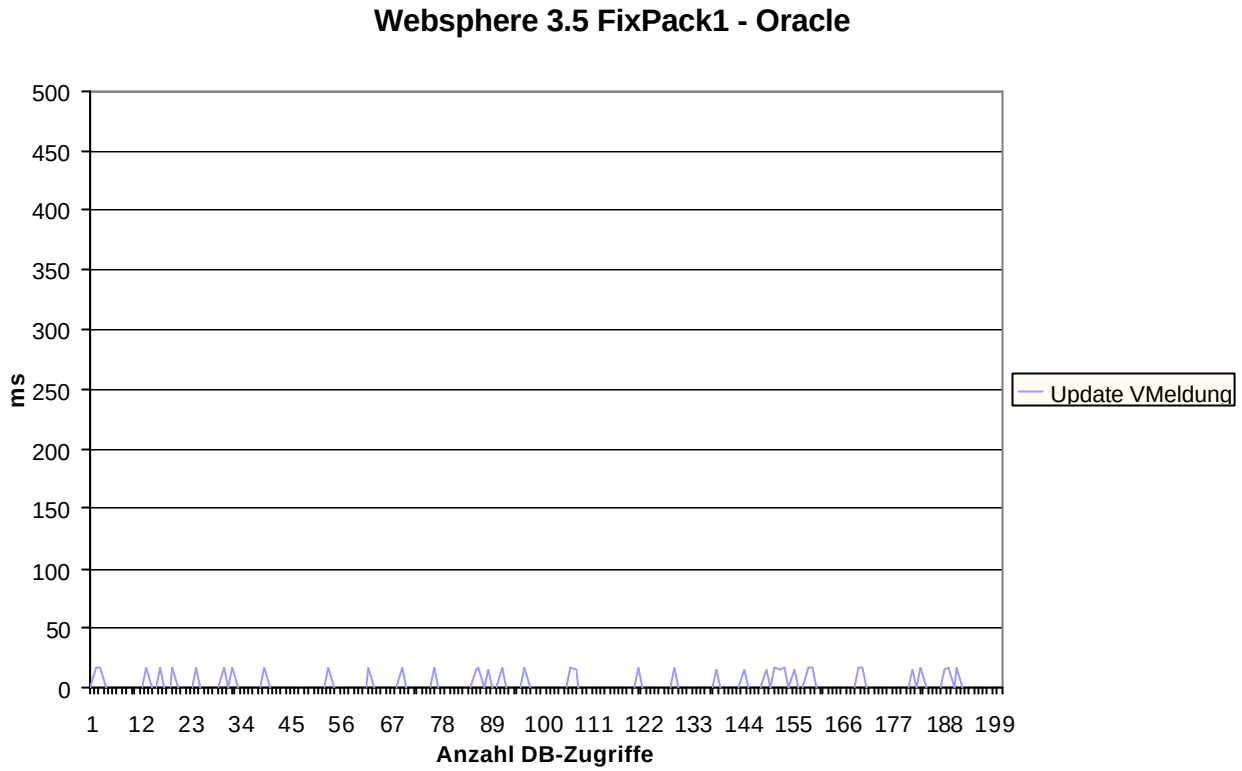
Beim Inprise Application Server konnte trotz Nutzung desselben Datenbanktreibers und Pooling ein Einschwingverhalten nicht abgestellt werden. Die Beans werden auch anders abgearbeitet (anderes Scheduling) was zu einer anderen Reihenfolge der Datenbankzugriffe führt.



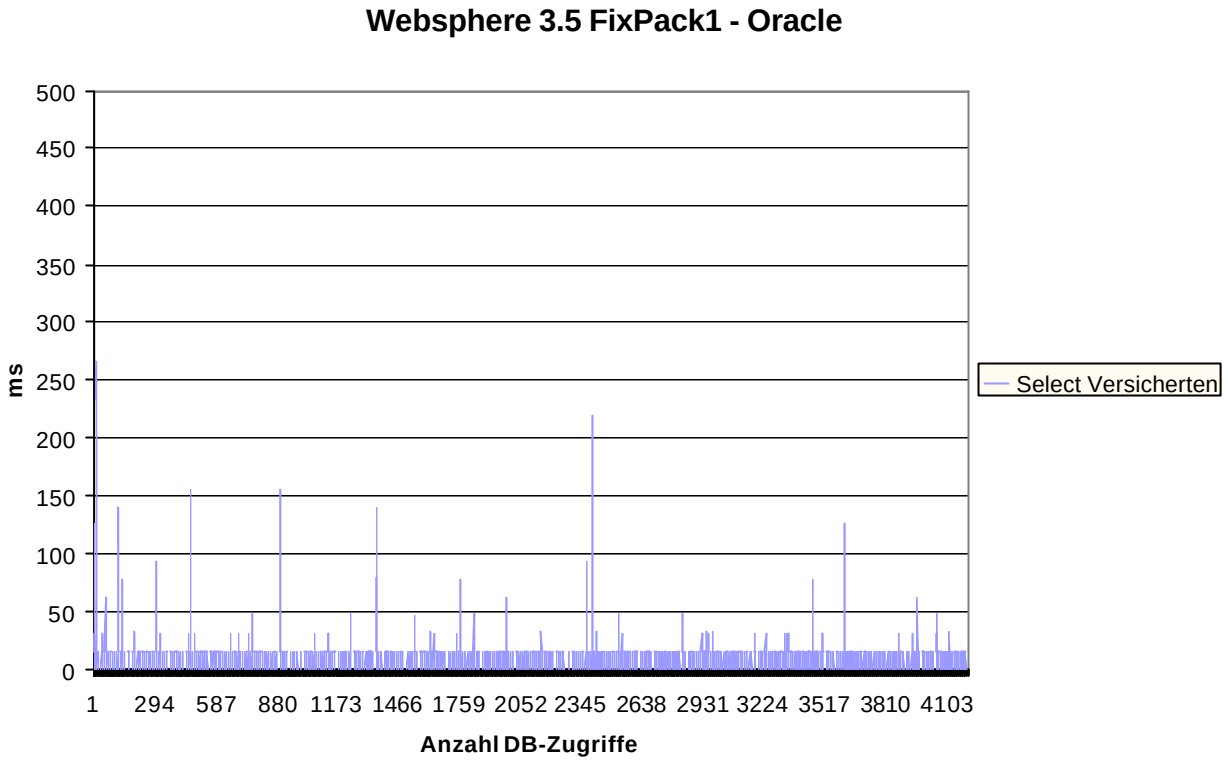
Bei der Abarbeitung der SELECT Anweisungen des Inprise Application Servers treten häufiger Spitzen auf, die allerdings nicht so hoch sind wie bei BEA. Auch hier ist wieder ein Einschwingverhalten zu beobachten.



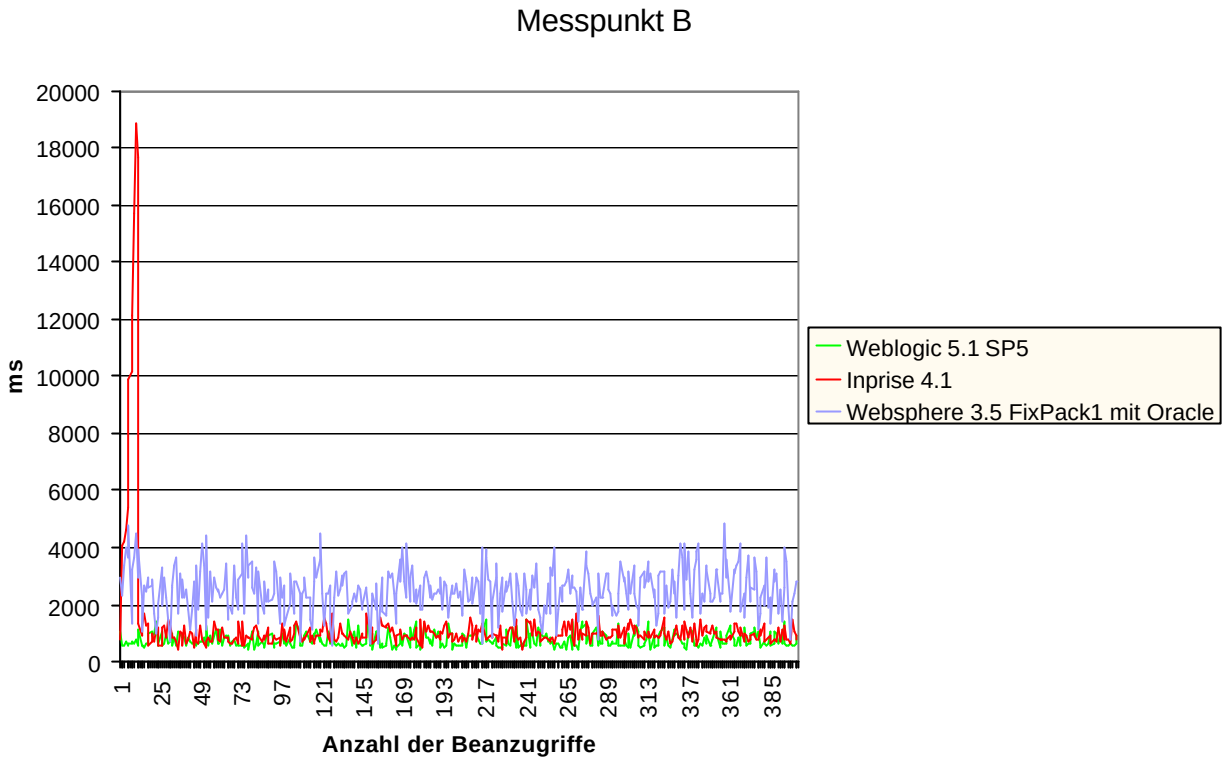
Websphere arbeitet die UPDATE-Anweisungen sehr gut ab. Es treten keine Spitzen auf, was sicherlich auch mit der etwas langsameren Abarbeitung der Beans zusammenhängt.



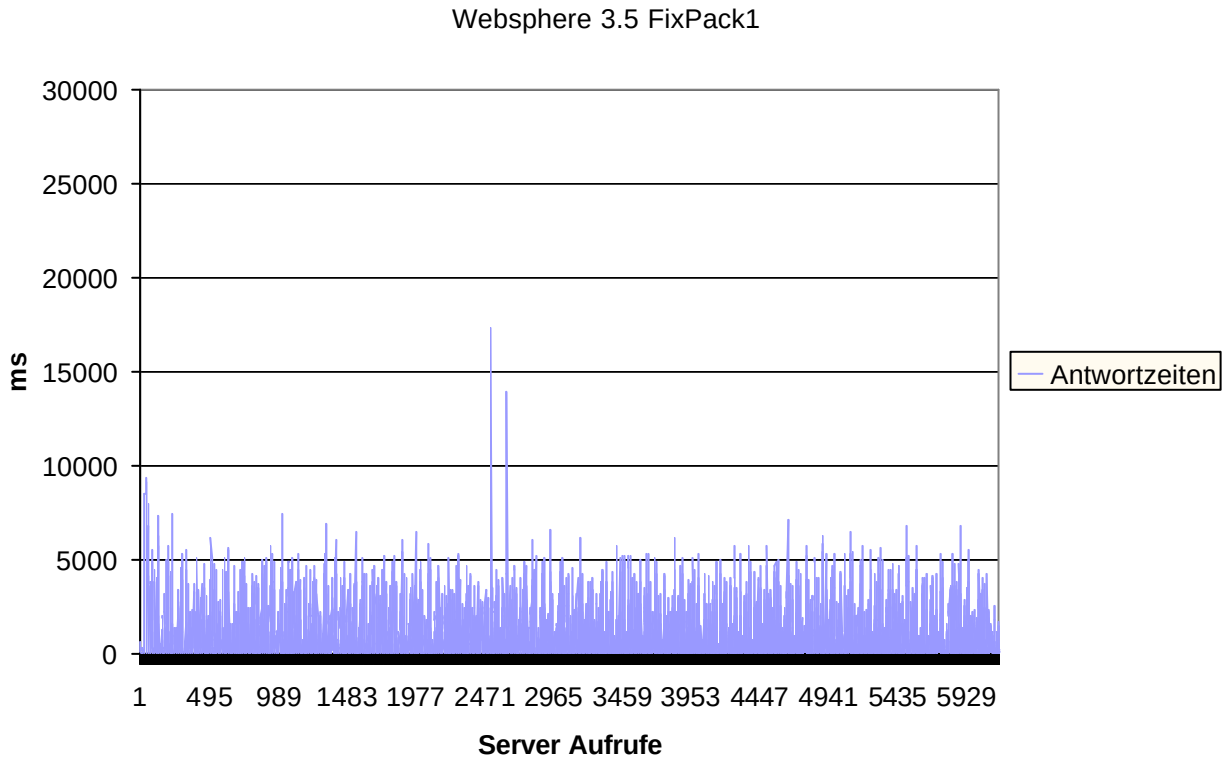
Bei den SELECT-Anweisungen treten ähnlich wie bei den anderen beiden Servern Spitzen auf. Insgesamt weisen die Server aber nur geringfügige Abweichungen bei dem Messpunkt C auf, was bei der gleichen Benutzung der Datenbank und des Treibers nicht verwundert. Die Abweichungen lassen sich durch das unterschiedliche Zugriffsverhalten und die unterschiedliche Abarbeitung der Beans, anderes Pooling etc. erklären.



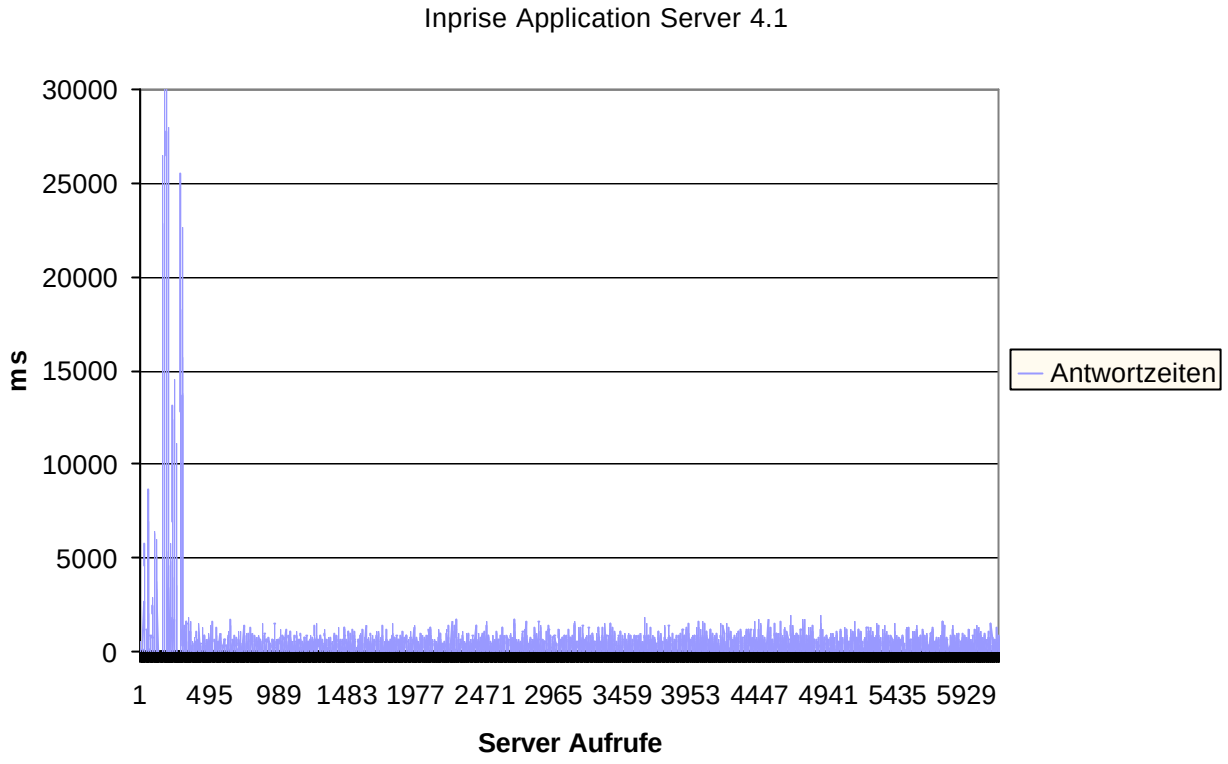
In diesem Diagramm sind die Ergebnisse des Messpunkts B zusammengefasst. BEA und Inprise liegen etwa auf demselben Niveau, wobei Inprise nur geringfügig mehr Zeit benötigt und das Einschwingverhalten auch hier zu erkennen ist. Websphere braucht hier schon deutlich mehr Zeit. Das bedeutet, dass IBM besonders in den Beans mehr Zeit für die Abarbeitung der Logik benötigt.



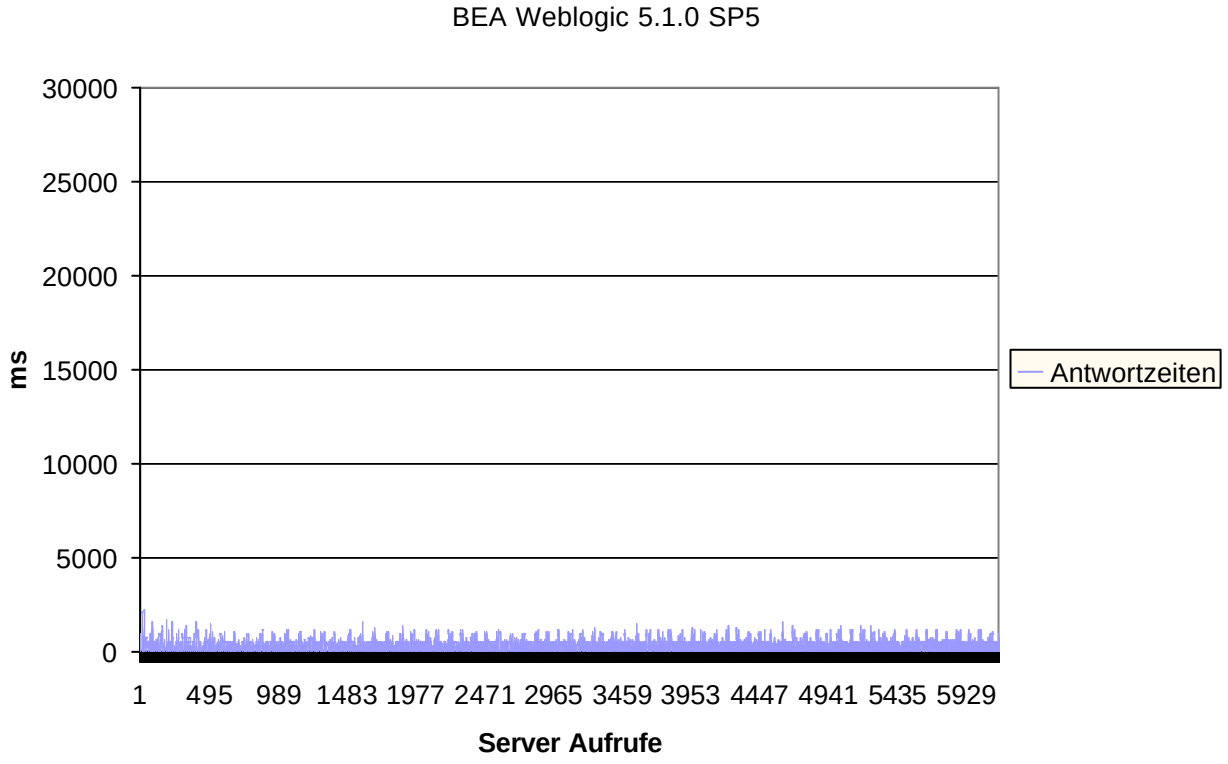
Die drei folgenden Diagramme zeigen die Antwortzeiten des Messpunkts A. IBM benötigt hier konstant mehr Zeit als die anderen beiden Server. Dabei ist der relative Unterschied in der Abarbeitung noch gestiegen, d.h. IBM braucht sowohl in den Beans, als auch in den Servlets, resp. JSPs mehr Zeit.



Bei Inprise ist die Abarbeitungszeit konstant sehr gering. Das Einschwingverhalten ist allerdings hier noch drastischer zu erkennen.

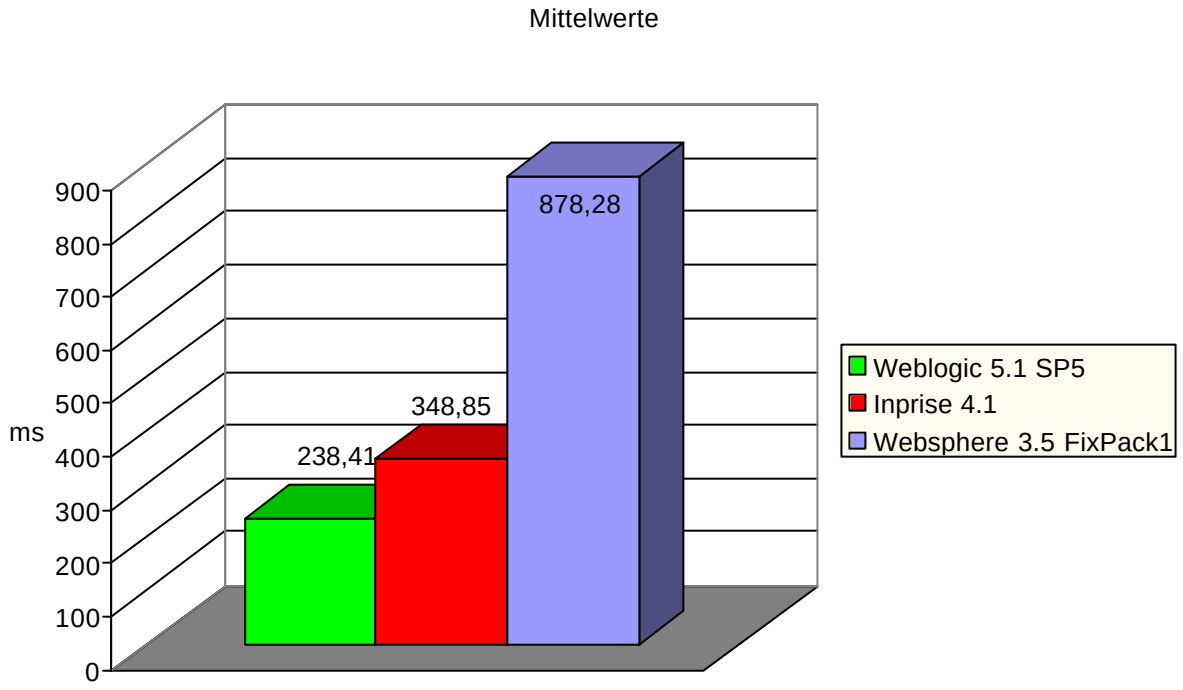


BEA kann genau wie Inprise mit einer gleichbleibend geringen Antwortzeit an Messpunkt A aufwarten.



Das folgende Diagramm fasst noch einmal die durchschnittlichen Messergebnisse des Tests zusammen. Dabei wurde der Mittelwert über alle Werte am Messpunkt A gebildet. Während Websphere ca. viermal soviel Zeit für den Test wie BEA benötigt, liegt Inprise etwa im Bereich von BEA. Auch wenn man das Einschwingverhalten von

Inprise mit berücksichtigt, schneidet Inprise in der Performance nur geringfügig schlechter ab. Die Messwerte sind in diesem Beispiel für 100 Nutzer durchgeführt worden. Leider treten auch bei dieser Anzahl schon Fehler bei der Ausführung auf den Servern auf. Inprise brach die Ausführung regelmäßig mit einer OutOfMemoryException ab, die auch mit Erhöhung des Speichers der JVM nicht umgangen werden konnte, was auf ein Speicherproblem innerhalb des Servers hindeutet. IBM konnte einige Transaktionen nicht abarbeiten, da es interne, nicht nachvollziehbare OSE Probleme gab. Die Tests wurden auf IBM und BEA auch mit 1000 Nutzern durchgeführt, wobei die Messergebnisse auf Grund der auftretenden Fehler leider nicht vergleichbar waren.



Ausblick

Da bei der Auswahl eines Applikationsservers nicht nur Performancewerte ausschlaggebend sind, sondern ein einfaches Deployment, die dafür benötigte Zeit und die Integration benötigter Entwicklungs- und Managementumgebungen eine ebenso große Rolle spielen, sollen die Arbeiten noch etwas in der Breite fortgesetzt werden. Dabei stehen dennoch weiterhin Performance-Aspekte im Vordergrund. Es ist außerdem geplant, die Tests auf andere Plattformen auszudehnen. So sollen Performancetests für Iona durchgeführt werden, sobald die Anbindung beliebiger Datenbanken möglich ist. Darüber hinaus sind Performancetests von iPlanet und Sybase angedacht. Außerdem soll das OR-Mapping Werkzeug Toplink eingesetzt werden, was insbesondere bei CMP relativ einfach durch Definition eines Descriptors möglich ist und ebenfalls durch den Einsatz von Caches Performancevorteile verspricht. Für das Produkt Javlin, welches ebenfalls Caches nutzt, soll der Aufwand evtl. Anpassungen bei EntityBeans genauer untersucht werden. In weiteren Beispielen sollen Unterschiede beim Einsatz von SessionBeans anstatt EntityBeans untersucht werden, da oft aus Performancegründen überlegt wird, SessionBeans anstatt EntityBeans einzusetzen. Dies resultiert vor allem aus der 1:1 Zuordnung der EntityBeans zu den Daten in der Datenbank. Der Performancegewinn beim Einsatz von Proxy-Objekten, welche auch in dem Versicherungsbeispiel massiv zum Einsatz kommen, soll ebenfalls noch mit genauen Performancezahlen versehen werden.

Literatur

1. Auswahl eines EJB-Applikations-Servers - Java-Spektrum 2/2000 S.12

2. Produktübersicht von EJB-basierten Applikations-Servern - Java-Spektrum 2/2000 S.18

3. Special Interest Group - Enterprise JavaBeans - http://www.mgm-edv.de/ejbsig/

4. Übersicht über EJB-Server - http://www.flashline.com/components/appservermatrix. jsp

5. Tutorium zur Programmierung von Enterprise-JavaBeans - Java Spektrum 3/2000 S. 47

6. Inprise Applikation Server - Java Magazin 2/1999 S.71

7. Read all about EJB 2.0 http://www.javaworld.com/javaworld/jw-06-2000/jw-0609-ejb.html

8. Richard Monson-Haefel: http://www.ejbnow.com

9. Enterprise JavaBean Persistence 101 – http://www.sdmagazine.com/uml/thinking/ s0008to.shtml

10. Enterprise JavaBean Persistence 201 – http://www.sdmagazine.com/features/ 2000/04/f3.shtml