Middleware Performance Benchmarking

---

bei EJB und .NET

Seminararbeit im Rahmen des Moduls
,,Komponentenbasierte Softwareentwicklung",
Jun.-Prof. Dr. Ralf H. Reussner im SS 2004
an der Carl-Von-Ossietzky Universität, Oldenburg

Klaus Krogmann

7. Juli 2004

Inhaltsverzeichnis

1 Abstract

Im vorliegenden Text wird COTS[1]-Middleware exemplarisch anhand der Beispiele .NET und J2EE / EJB betrachtet. Zum Vergleich der Leistungsfähigkeit unterschiedlicher COTS-Middleware-Plattformen werden üblicherweise Performance-Benchmarks herangezogen. Doch nicht alle Benchmarkingstechniken werden einem empirischen und wissenschaftlichen Anspruch vollkommen gerecht, wie das hier näher betrachtete Beispiel des „PetStore“ zeigen wird.

Welche Techniken für Benchmarks zu welchen Aussagen führen können, und welche Fallstricke sich beim Testen von insbesondere Middleware aufzeigen, wird ebenso beleuchtet, wie die generelle Problematik der Aussagekraft von Benchmarks unter Berücksichtigung von marktpolitischer Einflußnahme auf die, der Middleware gestellten, Szenarien.

Ziel dieses Textes ist nicht zuletzt ein Einblick in wissenschaftlich seriöse Benchmarking-Techniken und die immanenten Probleme durch die Komplexität moderner COTS-Middleware-Plattformen.

[1] Die Erläuterungen der im Abstract verwendeten Begriffe finden sie in den folgenden Kapiteln / der nächsten Seite.

2 Einleitung

Moderne Softwaresysteme werden tendenziell immer größer. Eine große Herausforderung liegt dabei in der Bewältigung der zunehmenden Komplexität solcher Systeme.

Große, verteilte Systeme lassen sich bezüglich ihrer Performance aber kaum im Vorfeld einschätzen[2]. Dennoch ist die Verarbeitung vieler gleichzeitiger Anfragen, auf Grund eines zunehmenden Grades der Verbreitung an netzwerkgestützen Applikationen, notwendiger denn je. Typische Szenarien verlangen neben der Behandlung von Zugriffs-Spitzen, hohen Transfer- und Benutzer-Volumina nach einer geringen durchschnittlichen Zugriffszeit und einer möglichst guten Skalierbarkeit des Systems.

Wie beispielsweise [LGL+02] anführt, besteht eine der Hauptan- und Herausforderungen für Softwarearchitekten im Entwurf von Soft-/Hardwareinfrastrukturen, die ein gefordertes Maß an Performance garantiert zusichern können. Das Problem stellt dabei nicht der Entwurf eines (speziellen) Prototypen dar, der einen bestimmten Fall erfüllt, sondern eine generische Lösung von Performanceanforderungen in einem komplexen System.

In der Literatur lassen sich diverse Methoden[3] zur Performancebestimmung von Systemen finden: UML-Diagramme mit Performance-Anotationen, „End-to-End"-Queuing oder „Layered Queuing Network". All diesen Modellen gemein ist der simulative oder analytische Charakter. Damit einher gehen Ungenauigkeiten der Ergebnisse, die ihren Grund u. a. in der abstrakten Betrachtung von COTS[4]-Middlewarekomponenten haben, denn im Gegensatz zum Benchmarking werden die Ergebnisse nicht validiert.

Tatsächlich wird die serverseitige Performance vieler Applikationen maßgeblich von vielen kleinen Faktoren auf der COTS-/Middlewareebene beeinflußt, weshalb einfache Tests häufig Gewissheit über die realen Performance-Eigenschaften geben können.

Welche genaue Methodik sich für empirische Tests und mathematische Modelle eignet, wird in diesem Papier unter anderem in Kapitel 4 aufgezeigt. Das entfernte Ziel ist also die Konstruktion eines brauchbaren Vorhersagemodells für Performanceeigenschaften, dass sich auf gesicherte reale Zahlen stützt, und Vor- und Nachteile bestimmter Architektureentscheidungen aufzeigen kann. Eine geeignete Test-Suite kann die Basisdaten hierfür liefern.

In welchem Maße Architekturentscheidungen und die konkrete Implementierung Einfluß auf die Performance ausüben, soll am Beispiel des „PetStore" (siehe Kapitel 5.1) dargestellt werden, das auf der einen Seite als Referenz durch Sun in Java (J2EE / EJB) implementiert wurde, im Rahmen der .NET-Bemühungen Microsofts auf der anderen Seite aber als Beleg für die Leistungsfähigkeit .NETs gelten soll.

Die Betrachtung von Performance-Benchmarking wird, wie folgt, dreiteilig vorgenommen:

[2] vgl. [LGL+02]
[3] siehe hierzu [LGL+02]
[4] COTS: Commercial Of The Shelf; COTS-Middleware z. B. in J2EE, .NET

1. Einführung mit Vorstellung und Definition von Middleware, EJB und .NET (siehe Kapitel 3).

2. Anforderungen an Benchmarking im Bereich von Middleware, Techniken und Einschränkungen (siehe Kapitel 4).

3. Konkreter Vergleich von .NET und EJB am Beispiel des PetStores (siehe Kapitel 5).

3 Einführung

3.1 Middleware

Glaubt man dem Werbespruch eines großen blauen Computerkonzerns, dann verbindet Middleware einfach alles.

Weniger umfassend läßt sich Middleware als eine Art „Dienstleister“[5] verstehen, der grundsätzlich unabhängigen Softwarekomponenten die Kommunikation untereinander, über eine Schnittstelle, ermöglicht. Middleware ermöglicht die Kommunikation auf einem hohen Niveau[6] und kümmert sich vornehmlich um:

- die Übertragung komplexerer Daten
- die Vermittlung von Funktionsaufrufen im Sinne von RPC[7]
- die Garantie von Transaktionssicherheit, die bei den ursprünglichen Basiskomponenten nicht sichergestellt werden könnte, wenn beispielsweise Transaktionen über die Grenzen einer einzelnen Komponente hinweg geschehen.

In typischen Szenarien erfolgt die Kommunikation der Komponenten untereinander dabei über Netzwerke und Protokolle wie TCP/IP und darauf basierenden Diensten wie HTTP, SOAP oder auch Web Services.

3.2 EJB

EJB steht als Abkürzung für „Enterprise Java Beans“ und bildet eines der zentralen Elemente der J2EE (Java 2 Enterprise Edition) Spezifikation. Die Spezifikation[8] wird im Java Community Process (JCP)[9] entwickelt und hat derzeit den Versionsstand 2.1 erreicht, in den vorrangig Erweiterungen im Bereich Web Services eingeflossen sind. Wie bei allen zentralen Java-Entwicklungen, steht Sun Microsystems als treibende Kraft hinter EJB / J2EE.

[5] vgl. [Wikb]
[6] bezogen auf das ISO/OSI-Schichtenmodell
[7] RPC: Remote Procedure Call, Aufruf „entfernter“ Methoden
[8] siehe [SM]
[9] Homepage des JCP: http://jcp.org/

Typischerweise werden EJBs in einer N-Tier-Architektur als Geschäftslogik ausgeführt und können nur in einem EJB-Container ablaufen, der wichtige Basisservices wie Transaktionsfähigkeit, Sicherheit und Persistenz anbietet. Der Container stellt dabei auch Mechanismen zur effizienten Ausführung mehrerer EJBs wie Thread-Verwaltung, Ressourcen- und Speicher-Management und Datenbankverbindungen zur Verfügung[10].

EJBs definieren als komponentenbasiertes Framework diverse Arten von Komponenten[11]:

- Entity Beans repräsentieren persistente Daten eines Systems, weshalb ihre Daten zumeist in Datenbanken gehalten werden. Bei dieser Form der Beans kann die Persistenz vom Bean-Entwickler selbst implementiert werden[12] oder über den EJB-Container angeboten werden[13].
- Session Beans werden in zustandsbehaftete „stateful“ und zustandslose „stateless“ Beans unterschieden. Zumeist dienen sie zur Modellierung von Interaktionen des Systems mit dem Nutzer.
- Eine weitere Form sind die Message Driven Beans, die für asynchrone Kommunikation zuständig sind. Vor allem Legacy-Systeme können damit angesprochen werden bzw. können darüber selber mit dem EJB-System in Kontakt treten.
- WebServices sind seit J2EE Version 1.4 verfügbar.

Für die Betrachtung von Performance-Eigenschaften sind jedoch vor allem die Implementierungen der J2EE interessant, da die bloße Spezifikation der EJB noch keine direkt meßbaren Performancewerte liefern kann. Wichtige Implementierungen sind hier:

- JBoss
- IBM WebSphere
- Sun Application Server 1.4
- Tomcat
- BEA WebLogic
- Geronimo

[10] siehe hierzu auch [LGL+02]
[11] vgl. [Wika]
[12] hier spricht man von „Bean Managed Persistence“, kurz BMP
[13] der Container übernimmt die Logik für die Datenhaltung; „Container Managed Persistence“, kurz CMP

3.3 .NET

Microsofts .NET-Plattform[14] ist eher eine komplette Familie von Produkten, denn eine Spezifikation, die zu Zwecken der Interoperabilität entworfen wurde[15]. Auch wenn es Open-Source-Implementierungen für ausgewählte Forschungseinrichtungen und Nicht-Microsoft-Plattformportierungen (z. B. Ximian) gibt, ist Microsoft der Hauptanbieter der .NET-Plattform.

Zur .NET-Plattform gehören auch Frameworks wie „Commerce Server“[16], die darauf abzielen, die Entwicklungszeiten auf speziellen Anwendungsgebieten möglichst klein zu halten, und somit die Kosten zu verringern, die durch eine effektive Implementierung von z. B. Containermanagement aber auch Performancevorteile gegenüber Eigenimplementierungen bringen können.

3.4 Ein grober Vergleich von EJB und .NET

.NET und J2EE / EJB haben beide eine gemeinsame Basis von unterstützten Standards. Beiden Plattformen liegt eine Multi-Tier-Architektur zu Grunde, die in die üblichen logischen Schichten, wie etwa Präsentation, Geschäftslogik und Datenhaltung unterteilt ist[17].

Die Frameworks von .NET und J2EE bringen beide Basis-Services wie Komponentenmanagement, Objekt-Persistenz, Transaktionen, Web-Services, Asynchrone Kommunikation, Eventbehandlungsunterstützung, Nachrichtenaustausch usw. mit[18]. Ebenso gemeinsam ist beiden Konzepten der Einsatz von Zwischencode statt platzformspezifischem Binärcode und eine darunter liegenden VM[19], die für Just-In-Time-Compilierung verantwortlich ist.

Es läßt sich konstatieren, dass vorhandenen Mechanismen aus den Frameworks in der Regel hochgradig performance-optimiert sind und Eigenimplementierungen nur schwerlich einen ähnlich hohen Grad erreichen dürften.

4 Benchmarking

Zum Benchmarking von Middleware sind vor allem die orthogonalen Kriterien „Performance“ und „Skalierbarkeit“ wichtig[20]. Wie in Abbildung 1 zu sehen, lassen sich damit grobe Klassifizierungen von COTS-Middleware vornehmen.

Weitere typische Meßgrößen sind: Transaktionen pro Zeiteinheit (Durchsatz), durchschnittliche Antwortzeit des Systems (Latenz), durchschnittliche Zeit bis zur Ergebnisausgabe durch das System, durchschnittliche Zeit bis zum Auftreten eines Fehlers (etwa Reliability), bei Echtzeitsystemen auch garantierte maximale Antwortzeit, etc.

[14] Das erste .NET-Äquivalent „Microsoft DNA and MTS“ wurde 1996 veröffentlicht.
[15] vgl. [You03]
[16] Framework zur Entwicklung von eCommerce-Applikationen
[17] siehe auch [You03]
[18] aus: [You03]
[19] VM: Virtual Machine
[20] vgl. auch [GTL+01], Seite 9

Daneben sind aber auch „unechte“ Benchmarkinggrößen denkbar: Entwicklungskosten, notwendige Codezeilen, Gesamtkosten zum Betrieb, Skalierungskosten, Managementkosten, usw.

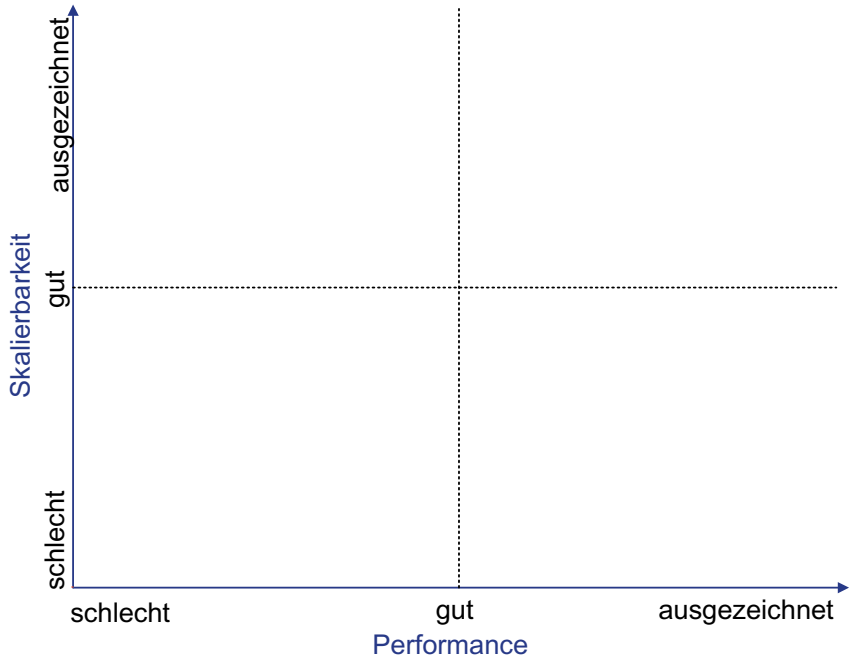

Abbildung 1: Benchmarking-Meßgrößen bei Middleware; Schema zur Einordnung von Middleware

4.1 Allgemeine Anforderungen an Benchmarks auf Middlewareebene

Im Weiteren sollte genau zwischen COTS-Middleware, das die Infrastruktur bildet und beispielsweise eine verteilte Umgebung für das Deployment zur Verfügung stellt, und den Komponenten der eigentlichen Applikation, die die Geschäftslogik und -abläufe bereitstellen, unterschieden werden. Da das komplette Lifecycle-Management, Ausführung, Sicherheitsprüfungen und z. B. Transaktionen[21] einer Komponente auf Applikationsebene von der COTS-Middleware-Umgebung übernommen wird, lässt sich eine solche Komponente nicht unabhängig von ihrer Umgebung untersuchen.

Da Middleware in den meisten Fällen auf Netzwerkkommunikation angewiesen ist, gilt es bei Performancetests diese relativ unzuverlässige (im Sinne von Reliabilitity) Ebene möglichst auszublenden. Daher verbietet sich insbesondere die Kommunikation über das Internet, es sei denn ausgerechnet die Reliability soll ermittelt werden.

[21] siehe hierzu auch 3.2, „Entity Beans“

4.2 Performance-Modell-Bildung

Das Verhalten der Applikationsebenen-Komponente hängt ganz offensichtlich maßgeblich von den darunter liegenden Schichten (also auch der COTS-Middleware) ab.

Durch diese starke Abhängigkeit untereinander wird die Abschätzung von unterschiedlichen Architekturvorteilen sehr komplex. Es ist nicht mehr trivial festzustellen, woher Performanceveränderungen stammen. Klassische Vorgehensweisen zur Performanceermittlung[22], bei denen z. B. eine Komponente für sich alleine getestet wird, sind nicht möglich.

Des weiteren ist der Quellcode von Komponenten zumeist nicht öffentlich zugänglich[23], so dass manuelle Inspektionen zu Performance-Analysezwecken nicht möglich sind.

Die Ermittlung der Performance über Tests ist aber auch von den konkret eingesetzen Produkten abhängig. [LGL+02] zeigt, wie stark Datendurchsatz und Skalierbarkeit am Beispiel von J2EE-Implementierungen schwanken. Bei .NET sind für das Framework derzeit die Microsoft-Implementierung und die freie Implementierung aus dem Mono-Projekt[24] in einer stabilen Version verfügbar. Dagegen ist das Spektrum verfügbarer Implementierungen auf der Java-Seite deutlich größer[25], das dadurch auch zu vielen Unterschieden bei der Performance führt.

Entsprechend sind nur jene Performance-Voraussagen aussagekräftig, die sich auf ein konkretes Produkt beziehen, bzw. genaue Unterscheidungen zwischen den Produkten vornehmen. Als Hardwareplattform sollte möglichst eine identische zum Einsatz kommen, um Performance-Effekte auf die eingesetzte Software begrenzen zu können.

Weitere Herausforderungen beim Test von Middleware liegen an den vielen Konfigurationsmöglichkeiten, die die COTS-Produkte überlicherweise bieten. Da für eine vollständige Erfassung von Performancedaten theoretisch alle Konfigurationen getestet werden müssten, ergeben sich durch die Kombinatorik zahllose zu testende Varianten. Dies führt zu langwierigen und kostenintensiven Prozeduren.

Ergebnisse der Performance-Benchmarks sollten abschließend genau erkennen lassen, welche Architekturentscheidung welche Auswirkungen auf die Performance haben, und wie sich Applikationen in der Vernetzung verhalten. Technische Variationspunkte müssen ebenso erfasst werden, um Softwarearchitekten und Entwicklern umgekehrt ein Framework zur Entscheidungsfindung zu liefern, womit weitere Softwareentwicklungen also wiederum von den Benchmarkingergebnissen profitieren können.

Zuletzt sollte das Verhalten der Komponente auf Applikationsebene[26] in Abhängigkeit von Eingabegrößen wie Geschäftslogikkomplexität, Eingabe-Transaktionsmix, Datenbankanforderungen etc. anhand des vorliegenden Modells

[22] vgl. [LGL+02]
[23] im Sinne von Blackbox-Komponenten
[24] Homepage siehe http://www.mono.org
[25] für eine kleine Übersicht siehe auch 3.2
[26] vgl. auch [LGL+02]

bestimmbar sein und aufzeigen, unter welchen Eingabeparametern ein bestimmtes Maß an Performance erreicht werden kann.

4.3 Testanforderungen

Wie [LGL+02][27] anführt, muss ein Benchmark-Test einigen Anforderungen genügen, um zuverlässige Aussagen über das Performanceverhalten machen zu können. Dazu zählen:

- Getestet werden müssen sowohl die Hauptkomponente, als auch alle weiteren optionalen Komponenten, die in einem typischen Anwendungsszenario vorkommen.
- Anhand der Ergebnisse muss nachvollziehbar sein, welchen Einfluss die Nutzung einer Teilkomponente auf die Gesamt-Performance des zu testenden Systems hat.[28]
- Naheliegenderweise sollte eine Test-Suite ökonomisch sinnvoll durchführbar sein. Insbesondere sollten die Ergebnisse schneller verfügbar sein, als eine neue Generation der zu testenden Software. Entsprechend schliessen sich mehrmonatige Testzyklen aus[29].

Wie bereits in Kapitel 4.2 erwähnt wurde, ist ein mit EJB erzieltes Ergebnis nicht *direkt* mit einem in .NET gewonnenen vergleichbar.

Um dennoch eine möglichst gute Vergleichbarkeit zu erreichen, muss für unterschiedliche Middleware-Technologien[30] das Design[31] der Test-Suite möglichst identisch sein. Die Unterscheidungen dürfen also nur auf Detailebene der Implementierung signifikant sein.

Daneben ist auch innerhalb einer COTS-Middleware-Spezifikation möglichst exakt die gleiche Test-Suite zu absolvieren[32]. Die auf Basis *einer* Spezifikation gewonnenen Ergebnisse lassen sich direkt vergleichen.

4.4 Benchmarkingtechniken

4.4.1 COTS-Middleware-Infrastruktur

Als Ausgangsbasis für weitere Benchmarks, sollte zunächst das Performanceverhalten der reinen COTS-Middleware ohne Einfluß von darauf aufsetzenden

[27] [LGL+02] macht Aussagen über Benchmarks bei J2EE / EJB, die allgemeine Konstruktion des Test-Szenarios läßt sich jedoch analog auf .NET (und ähnliche COTS-Middleware) übertragen.

[28] Vorausgesetzt, dass das System sich ohne diese Komponenten betreiben läßt. Unterscheidbar sollen also Variationspunkte der Konfiguration sein. [LGL+02] verweist an dieser Stelle als Beispiel auf Transaktionsfähigkeit, die über eine Komponente aktivierbar ist.

[29] Ausnahmen können benchmarkgestützte Auswertungen von z. B. speziellen Softwarearchitekturen bilden, da diese Erkenntnisse auch nach einem längeren Zeitraum noch von praktischer Relevanz sind.

[30] etwa J2EE oder .NET respektive COM+

[31] beispielsweise auch die Architektur der Test-Suite-Anwendung

[32] auf der gleichen COTS-Middleware-Spezifikation sollten sich die gleichen Test-Suiten durchführen lassen

Applikationen ermittelt werden[33]. Dadurch ist es möglich, später auf das Verhalten der eigentlichen Applikation zu schließen[34]. Dazu wird eine möglichst einfache Applikation ausgeführt, die alle Teile der Infrastruktur ausführt und jeweils spezifische Performance-Werte liefert.

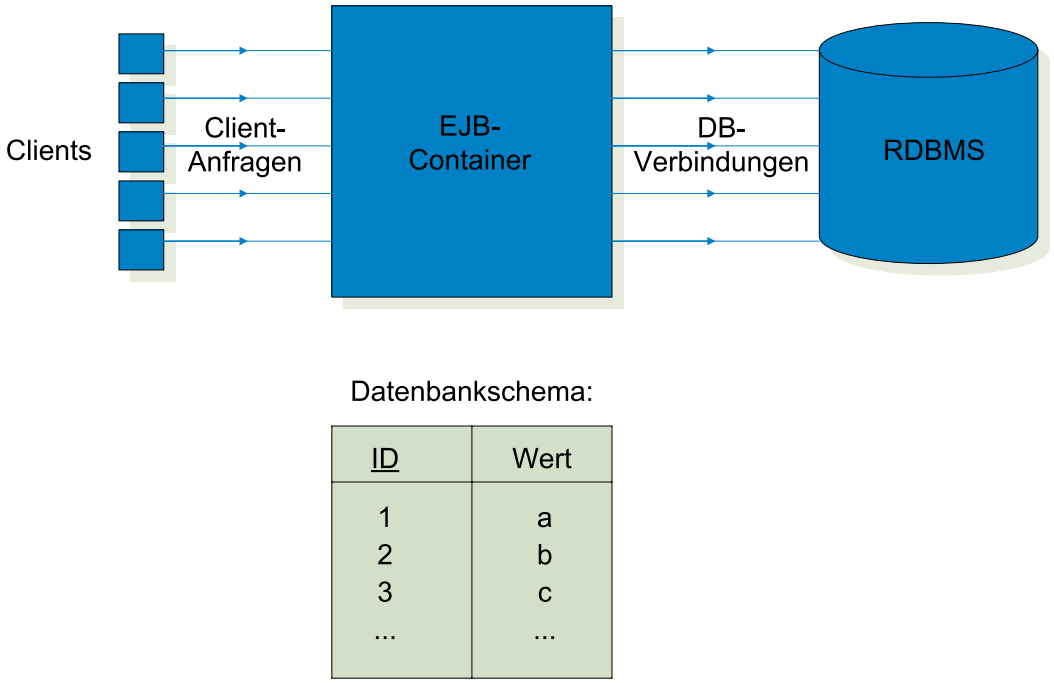

Abbildung 2: Grundsätzliche Architektur des Testablaufs mit der „Identitäts-Applikation“ zur Ermittlung der COTS-Middlewareperformance (aus „figure 5“, [LGL+02])

[LGL+02] schlägt hierzu einen Basis-Test-Fall vor, bei dem die Qualität der Komponenten-Infrastruktur (Server-Seite) quantitativ mit einer einfachen „Identitäts-Applikation[35]“ (Client-Seite) ermittelt wird, die folgenden Anforderungen genügt:

- Es gibt nur eine einzige Tabelle in einer relationalen Datenbank, deren Extension repräsentative Geschäftsdaten enthält.
- Das Datenbankschema sieht für die Tabelle nur die Attribute ID (unique) und einen zugehörigen Wert vor[36].
- Das Interface der einzigen Applikation (in der COTS-Middleware-Spezifikation implementiert) enthält nur Lese- / Schreib-Methoden.

[33] Dies setzt voraus, dass es *die eine* minimale COTS-Middleware-Konfiguration gibt, die ermittelt werden kann. In praktischen Tests (siehe auch Kaptiel 5.1) ist dies selten der Fall.

[34] Im Idealfall erhält man ein exaktes Profil der COTS-Middleware und kann die Einflüsse von den Applikationswerten subtrahieren.

[35] [LGL+02] spricht hier von „*identity* application“. [Bre03] betrachtet ebenfalls differenziert die „Größe“ der Benchmarkingsoftware.

[36] Key-Value-Pair

- Der lesende Zugriff erfolgt über die ID auf den zugehörigen Wert.
- Beim schreibenden Zugriff wird der über die ID bestimmte Wert imkrementiert.

Jede Client-Applikation[37] bekommt eine eindeutige ID zugeteilt, mit der sie auf den Server zugreift. Der Zugriff erfolgt dann entsprechend auf die Reihe der Datenbank, die über die ID zugewiesen wurde. Auf den Daten dieser Reihe werden dann zyklisch Lese- und Schreibzugriffe ausgeführt[38, 39].

Die Vorteile dieses Test-Szenarios lassen sich zum Bespiel im Vergleich mit Sun Microsystems „Ecperf Benchmark“[40] aufzeigen. Da Sun Microsystems Test komplexe Geschäftlogik enthält, lassen sich Performance-Effekte nicht auf die Applikation allein zurückführen, sondern können ebenso vom J2EE-Applikationsserver her rühren. In diesem Fall würde folglich lediglich die Gesamtperformance gemessen.

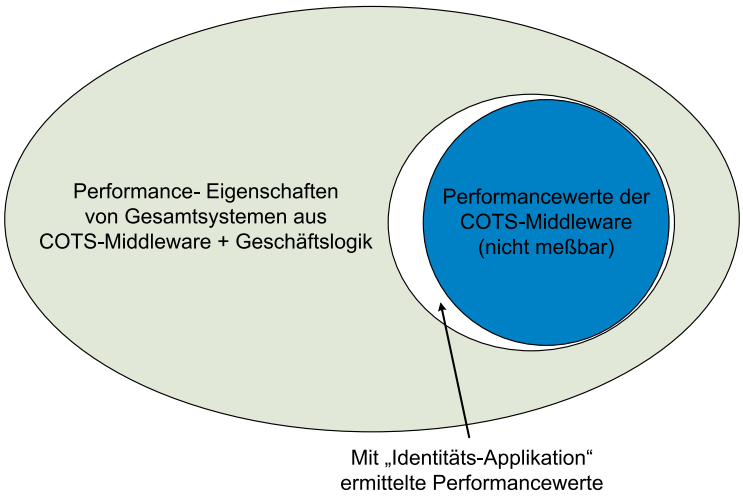

Abbildung 3: Teilmengendarstellung der Performanceeffekte nach Architekturebenen

Aber auch die von [LGL+02] vorgeschlagene Benchmarkingtechnik kann die Performance der COTS-Middlewarekomponente verständlicherweise nicht komplett isoliert ermitteln, sondern versucht die Einflüsse der Applikationsebene lediglich auf das mögliche Minimum zu reduzieren. Die mit der „Identitäts-Applikation“ ermittelten Werte stellen aber eine Teilmenge[41] der Performance-

[37] als Threads realisiert

[38] Der lesende Zugriff liefert den Wert zur ID zurück, beim schreibenden Zugriff wird der Wert um eins inkrementiert zurückgegeben, was einer minimalen Applikationslogik entspricht.

[39] Zur Überwachung des Performance-Tests kann z. B. Echtzeitmonitoring zum Einsatz kommen. Dadurch können Probleme bei einem langen Testlauf bereits frühzeitig erkannt und behoben, sowie der Fortschritt beobachtet werden.

[40] Performance Benchmark für EJB-Container

[41] Teilmengenbeziehung siehe Abbildung 3.

Effekte einer jeden Applikation dar[42], so dass bei allen größeren Applikationen ebenfalls mindestens diese Effekte auftreten.

4.4.2 Applikationsebene

Nachdem mit Hilfe der Identitäts-Applikation das Performanceverhalten der COTS-Middleware ermittelt wurde, kann der Basistest im nächsten Schritt um weitere Funktionalitäten und nicht-funktionale Eigenschaften[43] ergänzt werden. Durch die sukzessive Erweiterung läßt sich exakt feststellen, welche Komponente welche Performanceveränderungen bedingt. Die Aussagen für komplexe voneinander abhängige Erweiterungen lassen sich allerdings nicht zwingendermaßen auf eine Teilerweiterung zurückführen, da Komplettsysteme üblicherweise ein anderes Verhalten zeigen als Einzelteile[44].

Damit lassen sich bis jetzt die in Abbildung 4 dargestellten Abhängigkeiten feststellen.

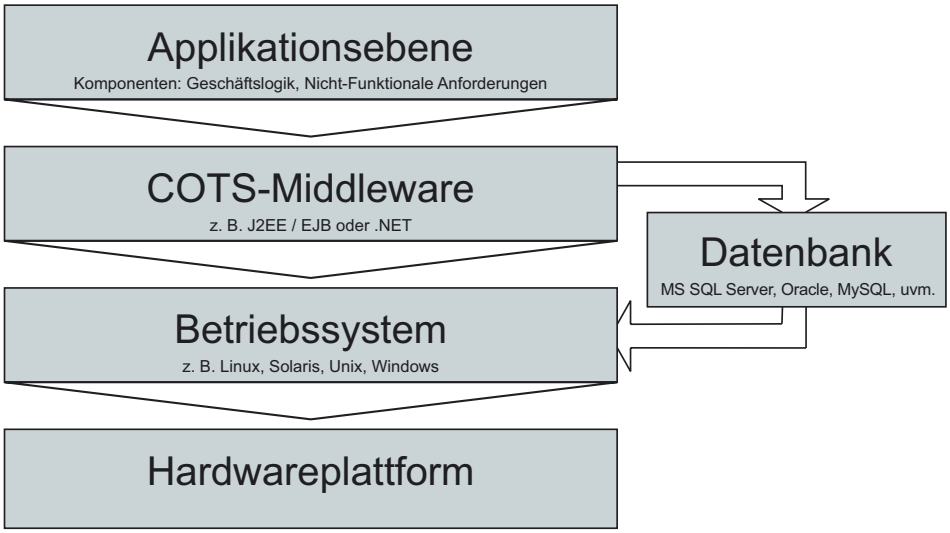

Abbildung 4: Relevante Ebenen der Testarchitektur

Abbildung 5 zeigt auf, welche der folgenden Variationen sich zur Untersuchung bezüglich ihrer Performanceauswirkungen heranziehen lassen.

Ergänzungen zu den Entscheidungsfindungskriterien aus Abbildung 5[45]:

1. Wenige Threads begrenzen die Serialisierungsperformance, wohingegen viele Threads übermäßig viele Ressourcen verbrauchen und dadurch die Organisation der Speicherverteilung der Applikation verlangsamen. Die Ermittlung des Optimums für ein bestimmtes Szenario erfolgt über Tests mit unterschiedlichen Konfigurationen.

[42] Man darf davon ausgehen, dass alle Applikationen mindestens die von der „Identitäts-Applikation“ aufgezeigten Charakteristika aufweist, da die „Identitäts-Applikation“ gerade als Minimalapplikation konstruiert wurde.

[43] z. B. Wartbarkeit, Nutzbarkeit, Sicherheit, Zuverlässigkeit

[44] im Sinne: „Ein System ist mehr als die Summer seiner Teile.“

[45] Die empirischen Testdaten für die EJB-Implementierung wurden in [LGL+02] veröffentlicht.

Parameter Typen	Beispielhafte Testparameter; Ermittlung des Optimums für diese Parameter	Ergänzungen in:
Systemparameter	Größe des Thread-Pools	1.
	Datenbankzugriffsorganisation	2.
	Größe Datenbankverbindungs-Pools	
	Cachegröße der Applikationskomponenten	
	Transaktionsstrategie	
Externe Eingaben	Client-Anfragelast	3.
	Client-Anfragefrequenz	3.
	Transaktionstypen	
	Transaktionsfrequenz	
Messbare / beobachtbare Parameter	Durchsatz (Transaktionen pro Sekunde)	3.
	Durchschnittliche Antwortzeit der Clients	3.
	Gesamtzeit der Clientanfragen	

Abbildung 5: Variationspunkte des Test-Szenarios (basierend auf „Table 2: Parameter Types and Sample Test Cases“ in [LGL+02])

2. Datenbankzugriffskonflikte[46] führen zu einer Verlangsamung der Zugriffszeiten. Um die genauen Auswirkungen zu ergründen, kann die Identitäts-Applikation angepasst werden.

   Dazu wird ein Maximum für die Anzahl der Datenbankzeilen festgelegt. Weiterhin greift jeder Client jetzt nicht mehr auf eine spezifische Datenbankreihe zu, sondern zufällig auf einen bestimmten Bereich.

   Je nach Wahl des Verhältnisses von Datenbankgesamtgröße zu Zugriffbereich für die einzelnen Clients wird die Zugriffskonfliktwahrscheinlichkeit steigen oder fallen[47].

3. Die Anfragelast durch Clients und die Frequenz der Anfragen zeigt neben einem „bloßen“ Performancewert bei Veränderung von Anfragelast und -frequenz auch die Skalierbarkeit von Middleware auf.

   Stellen mehr Clients ihre Anfragen oder eine konstante Anzahl Clients ihre Anfragen frequenter, als Server-Threads[48] im Pool zur Verfügung

[46] Zugriff durch mehrere Clients gleichzeitig auf die gleiche Datenbankzeile / die gleichen Teile der Datenbank

[47] $\frac{Datenbankgesamtgröße}{Clientzugriffsbereichsgröße} = x$; für größere x nimmt die Konfliktwahrscheinlichkeit ab.

[48] siehe hierzu auch Punkt 1.

stehen, müssen einige der Clients warten. Entsprechend verlängert sich die durchschnittliche Anfragezeit der Clients.

Trägt man die Zahl der Transaktionen pro Zeiteinheit (Durchsatz) auf der vertikalen Achse eines Diagramms ab und auf der Horizontalen die Zahl der zugreifenden Clients, sollte sich ein möglichst linearer Verlauf ergeben, damit man von einer guten Skalierbarkeit einer Middleware sprechen kann.

Es bleibt festzustellen, dass sich Performanceoptima von COTS-Middlewarekomponenten derzeit nur durch ausführliche, und dadurch teure, Tests ermitteln lassen. Zusätzlich erschweren kurze Produktlebenszyklen die Aussagefähigkeit von Performance-Benchmarks, da die gewonnenen Erkenntnisse jeweils nur für ein konkretes Produkt gültig sind.

5 Exemplarischer Performancevergleich

Im Folgenden soll für den exemplarischen Performancevergleich das „PetStore“-Beispiel herangezogen werden. Es liegt sowohl in Java (J2EE / EJB) als auch in .NET vor, und wurde in der Vergangenheit breit in der interessierten Öffentlichkeit diskutiert. Dabei haben die Entwickler der Java-Variante („The Middleware Company“[49]) detailliert ihre Entwurfsentscheidungen zur Java-Reimplementierung begründet[50], ebenso beleuchtet wurde aber auch das Test-Szenario, so dass dieses Beispiel einen guten Einblick in praktische Benchmarks auf Middleware-Ebene bietet.

Aus Gründen des begrenzten Umfang dieser Ausarbeitung, kann das Beispiel nicht komplett bis in alle Details untersucht werden. Insbesondere Abwägungen, ob „Stored Procedures“ o. ä. einen signifikanten Performancegewinn gebracht hätten, sollen hier nicht betrachtet werden. Statt dessen werden ausgewählte Aspekte des Testdesigns und der Vergleichbarkeit des Benchmarks betrachtet.

5.1 Das Beispiel ‘PetStore’

5.1.1 Hintergründe zum Benchmark

Der „PetStore“[51] wurde ursprünglich im Mai 2001 von Sun Microsystems als Beispielimplementierung für J2EE-Software entwickelt. Das Ziel war zunächst nicht die Implementierung eines Benchmarks, sondern der Entwurf einer Architektur-Vorlage mit offenem Quellcode für Entwickler von J2EE-Software, weshalb die Ausgangsversion nicht auf Performance optimiert war.

[49] „The Middleware Company“ ist eine Firma zur J2EE-Ausbildung und -Beratung; siehe auch [Dyc02]

[50] siehe u. a. [Com03]

[51] auch „Petshop“ bei .NET

Im Rahmen seines .NET-Engagements portierte Microsoft den PetStore auf C#[52]. Die Benchmarkingergebnisse von Microsoft sollten zeigen, dass .NET performanter als die ursprüngliche Java-Version sei. Da jedoch, wie bereits erwähnt, die ursprüngliche Java-Implentierung nicht auf Performance optimiert wurde, wurde die Aussagefähigkeit der Benchmarkergebnisse von Sun Microsystems, IBM und Oracle in Frage gestellt[53].

Im Oktober 2002 veröffentlichte „The Middleware Company“ (im Weiteren auch „TMC“) dann eine neue, auf Performance optimierte, J2EE-Version des PetStores („mPetStore“). Diese Version war rund 1700% schneller als die ursprüngliche Sun-Implementierung[54]. Im Vergleichstest mit .NET schnitt die Java-Version deutlich schlechter als das Microsoft-Pendant ab.

Diesem Test wurden jedoch mehrere Fehler und Fehlannahmen unterstellt[55]:

- Der ursprüngliche PetStore sei nicht performanceoptimiert gewesen. TMC hatte den Petshop jedoch komplett neu implementiert und auf Performance und Skalierbarkeit optimiert.

- Der Test sei fehlerhafterweise auf Intel-Hardware durchgeführt worden. Da Java auf Solaris und SPARC-Hardware optimiert wurde, waren hier auch die besten Ergebnisse erwartet worden. Wie in Kapitel 4 ausgeführt wurde, ist die gleiche Hardwareplattform (inklusive Betriebssystem[56]) jedoch eine Grundvoraussetzung für vergleichbare Benchmarkingwerte zwischen .NET und J2EE.

- Für J2EE wurde nicht die gleiche Datenbank wie für .NET verwendet. TMC begründet diesen Schritt mit der unterschiedlichen Performance. Da J2EE in Kombination mit Oracle über JDBC die performanteste Datenbankanbindung aufweist, wurde hier im Gegensatz zu .NET nicht der Microsoft SQL Server verwendet. Mit den in Kapitel 4 gewonnenen Erkenntnissen[57] bedeutet dies, dass sich die ermittelten Ergebnisse keinesfalls direkt vergleichen lassen. Aussagen können nur über den Gesamtkomplex, bestehend aus Datenbank, COTS-Middleware und Applikation gemacht werden. Es lassen sich also auch keine zuverlässigen Aussagen darüber machen, wie sich die zusätzliche Optimierung einer weiteren Teilkomponente der Applikation auswirken wird. Da im Zentrum der Betrachtung beim PetStore-Vergleich jedoch die Performance von .NET gegenüber J2EE für eine spezielle Applikation steht, lassen sich mangelnde Erkenntnisse über die Performance-Struktur der Applikation,

[52] Die Veröffentlichung der Portierung, hier „Petshop“ genannt, erfolgte im Oktober 2001. Die Architektur der derzeit aktuellen Version 3.x wird in [GLJD] erläutert.

[53] vgl. [kav]

[54] vgl. hierzu [Tea03b]

[55] Auswahl an Diskussionspunkten; vollständiger FAQ aus Herstellersicht (TMC) in [Tea03b]; eine öffentliche Diskussion im Forum von „The ServerSide.Com“ zum Thema unter [Lva]

[56] siehe auch Abbildung 4.

[57] Zur Einordnung der Datenbank in Abhängigkeiten des Test-Szenarios siehe auch Abbildung 4

bei ausreichender Berücksichtigung der verbleibenden Aussagefähigkeit, vertreten.

- Für die Tests auf Javaseite wurden unterschiedliche JDKs verwendet. Von Kritikern des Tests wurde der durchgehende Einsatz des JDK 1.4 angemahnt. Da TMC zum Zeitpunkt des Tests keine vollständig JDK 1.4 kompatiblen Applikationsserver zur Verfügung standen, wurde nur an allen Stellen, an denen die Möglichkeit bestand, das neuere JDK eingesetzt. Damit läßt sich festhalten, dass auch die Ergebnisse der J2EE-Tests untereinander nicht vollständig vergleichbar sein dürften.

- Ebenso wurde die Einführung eines veränderten Caches diskutiert. In der ursprünglichen Version wurden Daten auf der Applikationsebene gecached. Das Cachen in der User-Session sollte individuelle Anfragen besser handeln können. Da die Ausgangsversion aber bereits bei den Datenbankanfragen optimiert war, konnte nur ein unwesentlicher Performancegewinn festgestellt werden. Es zeigt sich also, dass nicht alle möglichen Stellschrauben in einem komplexen System den erwarteten Effekt erzielen. Vorhersagen sind durch die „unübersichtliche“ Konstruktion des Test-Szenarios nur eingeschränkt möglich.

Warum also der PetStore? Der Haupt-Java-Serverbenchmark SPECjAppServer2002[58] ist nur für J2EE spezifiziert. Daher sah Microsoft im PetStore eine gute Möglichkeit eine Vergleichbarkeitsapplikation für beide COTS-Middlewareplattformen zu entwickeln.

5.1.2 Aussagekraft des von TMC durchgeführten ‘PetStore’-Benchmarks

In der aufbrausenden Kritik[59] zeigt sich ein immanentes Problem von Benchmarks (besonders auf Middlewareebene). Ihre Aussagekraft ist stark von den gewählten Rahmenbedingunen abhängig. Durch die Vielzahl von Einstellungsmöglichkeiten der Software (bspw. bei der COTS-Middleware) ist es prinzipiell möglich ein Regelwerk für die Tests zu schaffen, dass je nach Wunsch die eine oder andere COTS-Middleware performanter aussehen läßt.

In einem prestigeträchtigen Vergleich wie dem von J2EE / EJB und .NET ist es daher nicht uninteressant, wer beispielsweise die finanziellen Mittel für den Test (ob direkt oder indirekt) zur Verfügung gestellt hat, da es ohnehin schon schwer ist, ein objektives Testszenario zu entwerfen, in dem komplexe Software von der COTS-Middleware bis hin zur Applikationssoftware in einem Schritt getestet werden soll.

Nicht zuletzt beeinflussen die detaillierten Implementierungsentscheidungen die Aussagekraft eines solchen Benchmarks. TMC hatte letztlich z. B. die Zahl der erzeugten Webseiten gemessen und daraus Rückschlüsse auf die Performance von J2EE / EJB bzw. .NET gezogen. Tatsächlich könnten in diesem

[58] früher auch bekannt als ECPerf; vgl. [Dyc02]
[59] vgl. etwa [Alm], [Lva]

Szenario aber nur seriös Aussagen über die Performance der COTS-Middleware im Zusammenspiel mit den konkret implementierten Beispielanwendungen gemacht werden, *nicht* jedoch über die COTS-Middleware alleine.

5.2 'PetStore' die 2.

5.2.1 Vergleichbarkeit der Performance-Tests

Im Juli 2003 veröffentlichte TMC eine zweite Performance-Studie[60], die zwar grundsätzlich sehr viel Ähnlichkeit mit der aus dem Oktober 2002 hatte, dessen Ergebnisse aber auf keinen Fall mit dem zweiten Performance-Test verglichen werden können, wie TMC in [Com03] auch selber anmerkt. Insgesamt waren Hardware[61], Clients, Applikationsserver, Datenbank, Betriebssystem[62] und der Test selber[63] unterschiedlich. Daneben gab es auch Unterschiede beim Einsatz verteilter Transaktionen und der Caching-Strategien. Da sich fast alle essentiellen Teile des Testszenarios geändert haben, verbietet sich ein seriöser Vergleich der ersten und zweiten Generation des Tests.

Für das eigentliche Ziel des Tests, den direkten Vergleich von J2EE / EJB und .NET, ist die fehlende Vergleichbarkeit der Tests untereinander jedoch irrelevant.

5.2.2 Testszenario

TMC hatte zur Minimierung von COTS-Middleware-Abhängigkeiten zunächst vier Implementierungen von J2EE getestet. Die zwei schnellsten wurden ausgewählt. Die Implementierung wurde zunächst in JSP/Servlet, JSP/EJB und .NET[64] durchgeführt. Bei den Java-Varianten stellte sich jedoch J2EE-EJB als die schnellere heraus. Bei den Applikationsserver war ein „App Server X“[65] die schnellere Wahl[66], dennoch wurde der Test ebenfalls mit einem zweiten „App Server Y“ durchgeführt. Die Tests wurden auf identischer Intel-Hardware durchgeführt.

Der Gesamttest gliederte sich in drei Hauptbereiche[67]:

- Test der Web-Applikation. Eine stetig steigende Zahl von Clients griff auf einen Server zu, auf dem eine typische Webanwendung (hier PetShop/mPetStore) lief.

- 24-Stunden Zuverlässigkeitstest[68]. Über einen Zeitraum von 24 Stunden

[60] Die gesamte Studie ist nachzulesen in [Tea03a].
[61] Zum Einsatz kam eine schnellere Hardware.
[62] Windows 2003 statt Windows 2000
[63] Das Test-Skript war grundsätzlich gleich, die „Bedenkzeit“ wurde jedoch rund halbiert.
[64] Für den Benchmark lieferte Microsoft die Version 2.0 des PetStore für .NET, die nach den Spezifikationen der TMC implementiert wurde. Siehe hierzu auch [kav].
[65] der Name wurde nicht weiter genannt, vgl. [Dyc02]
[66] vgl. [Tea]
[67] vgl. [Tea]
[68] im Sinne von Reliabitity

lief ein transaktionsintensives Skript gegen die Web-Applikation und simulierte eine konstante, durchgehend spitzenbelastende Nutzerlast.

- Web-Services Test. Die Performance des Applikationsserver beim bereitstellen von Basis-Webdiensten via SOAP 1.1 wurde getested.

Als zusätzliche Testparameter wurden die Codezeilen und die Gesamtkosten verglichen. Diese Parameter haben jedoch keinen direkten Einfluß auf die Performance und lassen die Testergebnisse unschärfer werden, da hier zwei Klassen von Vergleichbarkeitskriterien vermischt werden.

5.2.3 Testergebnisse

„.NET ist also schneller (und besser) als J2EE“ lautet überspitzt das Ergebnis der TMC.

Beim zweiten Test hat TMC sich bemüht häufige Kritikpunkte an dem ersten Test auszuräumen und insbesondere den Code der Java-Version noch weiter zu optimieren. Dennoch bleiben auch beim zweiten Test grundsätzliche Einschränkungen zur Aussagefähigkeit, die in Kaptitel 5.1.2 genannt wurden, erhalten[69].

5.3 Diskussion

In beiden Fällen wurde COTS-Middleware in einem Schritt mit der Applikationsebene getest. Aussagen sind folglich nur für Gesamtsysteme möglich, insbesondere nicht im Sinne „.NET ist schneller als J2EE / Java“. Es sind immer nur „gleichwertige“ Tests (insbesondere nicht 1. + 2. Version des PetStore) untereinander vergleichbar. Im Speziellen wäre nur eine Vergleichbarkeit innerhalb von .NET und J2EE des gleichen Tests mit oben genannten Einschränkungen möglich.

Daher sind, ob der von TMC verlautbarten Ergebnisse, öffentliche Diskussionen nicht verwunderlich. So erklärte Rickard Öberg[70] die Benchmarkergebnisse für „nicht glaubwürdig“[71] und führte, wie in der öffentlichen Diskussion am weitesten verbreitet, die mangelnde Optimierung der Java-Version und die komplexere Architektur der Java-Version ins Feld, die der .NET-Version Performanz verschaffe. Dabei unterstellte er „der Middleware Company sogar, die Ergebnisse absichtlich zugunsten von Microsoft beeinflußt zu haben“[72].

6 Fazit

Aus den gewonnenen Erkenntnissen lassen sich drei Paradigmen ableiten:

[69] weshalb TMC beim zweiten Test auch bewußt von „Performance Study“ statt „Benchmark“ spricht; vgl. [Tea03a]

[70] schwedischer Entwickler des J2EE-Servers „JBoss“

[71] vgl. [kav]

[72] aus [kav]

- Aussagen der Art „Produkt A ist besser als Produkt B“, bezogen auf Benchmarking von (handelsüblichen) COTS-Middlewareplattformen, sind unseriös[73].

  Selbst COTS-Middlewareplattformen alleine sind bereits so komplex, dass es nicht möglich ist, eine als die klar performantere zu ermitteln. Aussagen lassen sich nur für exakt spezifizierte Rahmenbedingungen machen.

- Die „Identitäts-Applikation“ erscheint ein geeignetes Mittel zu sein, um COTS-Middleware-Performance zu analysieren und ein Vorhersagemodell zu erschaffen.

  Durch die vielfältigen Einstellungsmöglichkeiten von COTS-Middleware dürfte es in der Praxis schwer fallen, *die eine* Minimalkonfiguration zu ermitteln, deren Ausgangsbasis die „Identitäts-Applikation“ ist. Dennoch sollten Performance-Test möglichst feingranular oberhalb der identischen Testplattform[74] vorgehen.

- Komplexe Tests auf Applikationenebene sind ungeeignet um die Performance von COTS-Middleware zu analysieren.

  Durch vielfältige gegenseitige Abhängigkeiten von Applikationebene und COTS-Middleware lassen sich Performanceeffekte nicht exakt auf eine Teilkomponente zurückführen. Außerdem lassen sich mit dieser Methodik keine präzisen Schemata erzeugen, die Performance-Vorhersage über eine COTS-Middleware vor und nach Hinzufügen einer Komponente ermöglichen.

Wie das Beispiel des PetStore bei Java und .NET zeigt, sind Benchmarkingergebnisse grundsätzlich mit Vorsicht zu genießen. Erst durch die Offenlegung des genauen Test-Szenarios mit allen eingesetzten Komponenten, dem Quellcode des Tests und der verwendeten Parameter, läßt sich die Aussagefähigkeit eines Test beurteilen - oder auch über was der Benchmark tatsächlich Performance-Analysen zuläßt.

Nicht zuletzt durch die einfache Manipulationsfähigkeit ob der vielen Einfluß nehmenden Parameter besteht die Gefahr, dass Benchmarks (markt-) politisch mißbraucht werden. Daher ist es notwendig, Benchmarkergebnisse stets kritisch zu hinterfragen und festzustellen, was tatsächlich gemessen wurde / werden konnte. Es gibt keinen goldenen Weg, der immer zu dem richtigen Ergebnis führt, wenngleich die in Kapitel 4 vorgeschlagene empirische Vorgehensweise ein Schritt in die richtige Richtung ist.

[73] vgl. Kapitel 5
[74] vgl. Kapitel 4.4.1

7 Referenzen

Der Text wurde unter Verwendung der folgenden Literatur erstellt.

Literatur

[Alm] Dion Almaer. *Another Review of „The Petstore Revisited: J2EE vs .NET Application Server Performance Benchmark“*. O'Reilly, http://www.oreillynet.com/pub/wlg/2240. Retrieved 2004-05-18.

[BCG+02] Paul Brebner, Shiping Chen, Ian Gorton, Jeffrey Gosper, Lei Hu, Anna Liu, Doug Palmer, and Shuping Ran. Evaluating j2ee application servers. Technical report, CSIRO Mathematical and Information Sciences, Macquarie University, Australia, 2002.

[BG02] Paul Brebner and Ian Gorton. Evaluation j2ee application servers. Technical report, CSIRO Mathematical and Information Sciences, Macquarie University, Australia, 2002.

[Bre03] Paul Brebner. The impact of object oriented characteristics of middleware benchmarks. Technical report, CSIRO ICT Research Centre, ANU Canberra, Australia, 2003.

[CCE+03] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. Performance comparison of middleware architectures for generating dynamic web content. Technical report, INRIA, Projet Sardes, France; Rice University, USA; Ecole Polytechnique Fédérale de Lausanne, Switzerland, http://www.cs.rice.edu/ anupamc/publications/mid2003.pdf, 2003. Retrieved 2004-05-18.

[CGT02] Charles Chung, Jeffrey Gosper, and Phong Tran. Evaluation of eai/bpm series vitria businessware 3.1.7. Technical report, CSIRO MATHEMATICAL AND INFORMATION SCIENCES, 2002.

[Com03] The Middleware Company. Comparing the results of the july 2003 performance study with the results of the october 2002 performance study. Technical report, The Middleware Company, July 2003.

[Dyc02] Timothy Dyck. *There Ain't No Business Like Benchmarking*. eWeek, http://www.eweek.com/article/2/0,1759,798232,00.asp, December 2002. Retrieved 2004-05-18.

[GLJD] Microsoft Corporation Gregory Leake and Inc. James Duff, Vertigo Software. *Microsoft .NET Pet Shop 3.x: Design Patterns and Architecture of the .NET Pet Shop*. Microsoft Corporation, Vertigo Software, Inc., http://www.microsoft.com/indonesia/msdn/PetShop3x.asp. Retrieved 2004-05-18.

[GTL+00] Ian Gorton, Phong Tran, Anna Liu, Shiping Chen, and Paul Greenfield. Evaluating enterprise middleware technologies. Technical report, CSIRO Mathematical and Information Sciences, Macquarie University, Australia, 2000.

[GTL+01] Ian Gorton, Phong Tran, Anna Liu, Shiping Chen, and Paul Greenfield. Evaluating enterprise middleware technologies (com+). Technical report, CSIRO Mathematical and Information Sciences, Macquarie University, Australia, 2001.

[kav] kav. *Performance-Vergleich: .NET schlägt J2EE [Update].* heise online, http://www.heise.de/newsticker/meldung/32003. Retrieved 2004-05-18.

[LGL+02] Yan Liu, Ian Gorton, Anna Liu, Ning Jiang, and Shiping Chen. Designing a test suite for empirically-based middleware performance prediction. Technical report, Basser Department of Computer Science, University of Sydney, Australia, 2002.

[LIGH01] Anna Liu, Ian, Gorton, and Lei Hu. Evaluating bea systems application server technology. Technical report, CSIRO Mathematical and Information Sciences, Macquarie University, Australia, July 2001.

[Lva] Greg Leake and various authors. *.NET vs. Oracle's Java PetShop Benchmark Round III.* The ServerSide, http://www.theserverside.com/news/thread.tss?thread_id=13700. Retrieved 2004-05-18.

[Mic] Microsoft. *Using .NET to Implement Sun Microsystems' Java Pet Store J2EE BluePrint Application.* Microsoft, MSDN, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdasamppet2.asp. Retrieved 2004-05-18.

[SM] Inc. Sun Microsystems. *EJB 2.1 specification.* Sun Microsystems, Inc., http://java.sun.com/products/ejb/docs.html. Retrieved 2004-06-02.

[Tea] The Middleware Company Research Team. *J2EE and .NET Application Server and Web Services Performance Study.* The Middleware Company, http://www.middlewareresearch.com/endeavors/artifacts/030730J2EEDOTNET/endeavor.jsp. Retrieved 2004-05-18.

[Tea03a] Research Team. J2ee and .net (reloaded) yet another performance study. Technical report, The Middleware Company, June 2003.

[Tea03b] The Middleware Company Research Team. Summary of october 2002 study feedback; we incorporated into the mpetstore startingpoint. Technical report, The Middleware Company, 2003.

[TG02] Phong Tran and Jeffrey Gosper. Eai/bpm evaluation series ibm websphere mq workflow v3.3.2 + eai suite. Technical report, CSIRO MATHEMATICAL AND INFORMATION SCIENCES, 2002.

[Wika] Wikipedia. *Enterprise Java Beans*. Wikipedia, die freie Enzyklopädie, http://de.wikipedia.org/wiki/EJB. Retrieved 2004-05-18.

[Wikb] Wikipedia. *Middleware*. Wikipedia, die freie Enzyklopädie, http://de.wikipedia.org/wiki/Middleware. Retrieved 2004-05-18.

[Woo03] Jongwook Woo. The comparison of j2ee and .net for e-business; the technical report (hipic-10292003) of high-performance information computing center at california state university, los angeles. Technical report, Computer Information Systems Department, California State University, Los Angeles, http://www.calstatela.edu/faculty/jwoo5/publications/hipic-techreport-102912003.pdf, 2003. Retrieved 2004-05-18.

[You03] Sameh Younis. *J2EE vs. .NET- An Executive Look*. http://www.netreverie.com/dotnet/J2EE_vs_.NET-An_Executive_Look.asp, September 2003. Retrieved 2004-05-18.