4. Lastmodellierung und -analyse

Last beeinflusst in starkem Maße die Systemleistung

$\Rightarrow$ erster Schritt der Leistungsanalyse

Verständnis der Last

Beschreibung der Systemlast i.a. in Form

eines mathematischen Modells

Zur Modellerstellung werden Daten benötigt,

diese müssen erst erhoben werden

Zur Kapazitätsplanung werden Aussagen über

die zukünftige Last benötigt, diese muss

aus der heutigen Last abgeleitet werden

Kapitelgliederung

4.1 Charakterisierung der Systemlast

Lastcharakterisierung und -parameter hängen
vom Ziel der Analyse ab

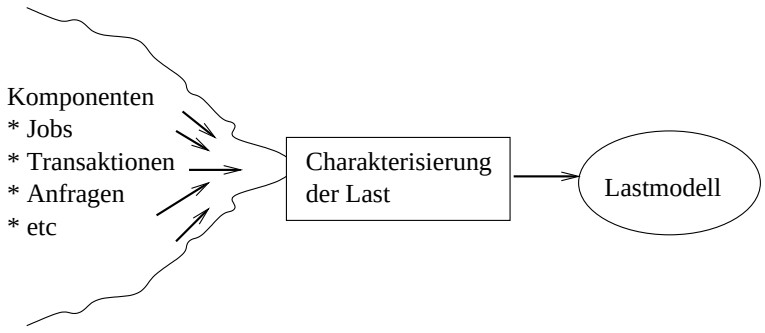
z.B. Analyse eines Web Servers

a) Untersuchung der Auswirkung eines Proxy Caches
Zugriffsfrequenzen auf Seiten,
Lokalität der Zugriffe, Größe der Dokumente

b) Untersuchung der Auswirkung einer neuen CPU
Lastbeschreibung enthält CPU Zeit pro Zugriff,
Anzahl E/A-Operationen pro Zugriff

Schritte der Lastcharakterisierung:

1. Spezifikation der Ziele
2. Auswahl der Parameter
3. Messung am System zur Datenermittlung
4. Analyse und Reduktion der Daten
5. Konstruktion des Lastmodells



4.1.1 Erste Schritte

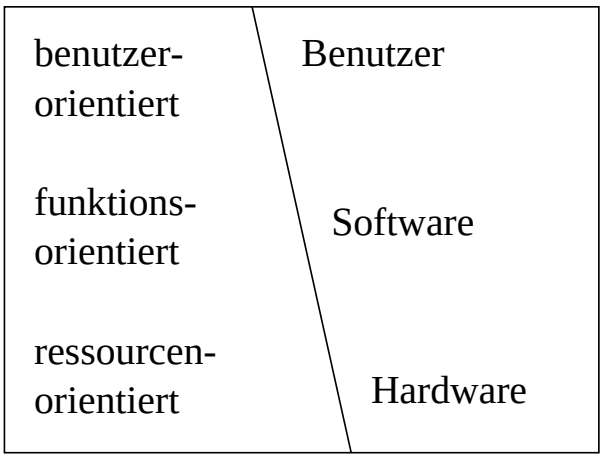
Identifikation der Basiskomponenten

Basiskomponenten sind generische Einheiten

der im System auftretenden Last

z.B. Transaktionen, Jobs, HTTP Anfragen, ...

Last kann auf verschiedenen Ebenen beschrieben werden



in Abhängigkeit vom Analyseziel

und im Zusammenhang mit dem Systemmodell

- benutzerorientiert: Lastkomponenten sind Programme oder Befehle des Benutzers
- funktionsorientiert: Lastkomponenten sind Befehle im Programm
- ressourcenorientiert: Lastkomponenten sind Funktionen der Hardware

Ein Beispiel

Messung von 10 Anfragen an einen Web Server

nur CPU Zeiten, E/A Zeiten und Antwortzeiten gemessen

Anfrage Nr.	CPU Zeit	E/A Zeit	Antwortzeit
1	0.095	0.04	0.071
2	0.0130	0.11	0.145
3	0.0155	0.12	0.156
4	0.0088	0.04	0.065
5	0.0111	0.09	0.114
6	0.0171	0.14	0.163
7	0.2170	1.20	4.380
8	0.0129	0.12	0.151
9	0.0091	0.05	0.063
10	0.0170	0.14	0.189
Durchschnitt	0.0331	0.205	0.550

Wie kann diese Last im Modell gefasst werden?

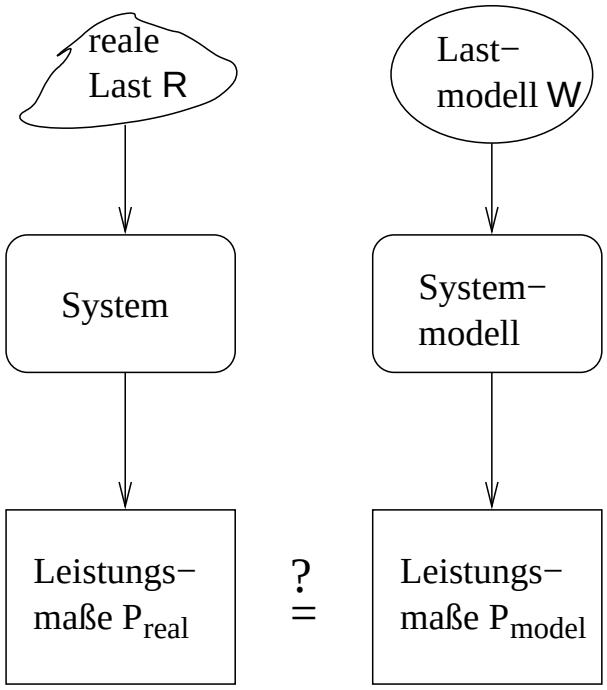
Einfachste Möglichkeit: Repräsentation durch Mittelwerte

also CPU Zeit 0.0331, E/A Zeit 0.205 Sekunden pro Anfrage

Wie repräsentativ ist diese Darstellung?

Wann ist eine Last repräsentativ?

Sei R die reale Last und W das Lastmodell



W ist eine perfekte Repräsentation von R ,

falls $P_{real} = P_{model}$

Also Qualität der Lastdarstellung immer

in Abhängigkeit vom Leistungsmaß

Im Beispiel:

Lastdefinition ist adäquat falls Antwortzeiten

genau genug im Modell wiedergegeben werden

Dies hängt sicherlich auch davon ab, ob

- nur die mittlere Antwortzeit
- die Verteilung der Antwortzeiten

betrachtet wird

Falls Mittelwertbildung nicht repräsentativ,

da reale Antwortzeiten zu stark schwanken

$\Rightarrow$ Definition von Lastklassen,

um Schwankungen in der Last zu erfassen

Einteilung in Klassen

- klein: CPU Zeit zwischen 0.0001 und 0.0099, E/A-Zeiten zwischen 0.0 und 0.05
- mittel: CPU Zeit zwischen 0.01 und 0.03, E/A-Zeiten zwischen 0.06 und 0.14
- groß: CPU Zeit größer 0.03, E/A-Zeiten größer 0.14

Vorsicht: Einteilung nicht immer so einfach,

reale Lasten enthalten mehr als 10 Elemente

sind nicht so einfach Klassen zuzuteilen

(später mehr dazu)

Dies liefert folgendes Lastmodell

Typ	CPU Zeit	E/A-Zeit	Komponenten
klein	0.0091	0.04	3
mittel	0.0144	0.12	6
groß	0.2170	1.20	1

Deutlich bessere Darstellung der Schwankungen

in den Anforderungen

Lastmodelle

Man unterscheidet

- ausführbare Lastmodelle
- nicht ausführbare Lastemodelle
  (nur Beschreibung der Ressourcenbelastung)

Lastmodelle können

- zur Messung von Systemleistung
- als Eingabe von Simulatoren
- zur Parameterbelegung analytischer Modelle

verwendet werden

Dadurch ergeben sich Einschränkungen bzgl. der Darstellung

Wir benötigen Daten

zur Parametrisierung analytischer Modelle

Also keine

- ausführbaren Modelle
- Verteilungen

sondern

- mittleren Bedienbedarf an Ressourcen
- mittlere Besuchshäufigkeiten
- Ankunftszeiten

durch Einführung von Klassen Variabilität abbilden

4.1.2 Methodik der Lastcharakterisierung

Identifikation der Basiskomponenten

- Abhängig vom System und Analyseziel
- Liefert eine Menge von unterschiedlichen Komponenten mit jeweils individuellem Verhalten/individuellen Anforderungen

Auswahl der Parameter

Für jede Komponente Parameter definieren

Man unterscheidet

- Lastintensitäten: Ankunftsrate, Anzahl Clients oder Prozesse
- Bedienanforderungen: beschrieben durch
  K-Tupel $(D_{1,r}, \ldots, D_{K,r})$ mit $D_{i,r}$
  Bedienbedarf von Komponente r an Ressource i

Datenerhebung

- Zeitfenster zur Datenerhebung (Messung) festlegen
- Beobachten und messen des Systems während des Zeitfensters
- Bestimmung der Systemparameter aus den gemessenen Daten (u.U. indirekt z.B über Little)

später mehr zur Datenerhebung

Partitionierung der Last

in der Regel besteht Last aus heterogenen Komponenten

z.B. Anfragen an WWW Servern

- vom kleinen HTML Dokument bis
- zum ganzen Video

Mittelwertbildung über heterogene Anfragen

- führt zu sehr groben und oft ungenauen Modellen
- macht die Lastvorhersage unmöglich

Ziel der Partitionierungstechniken:

Klasseneinteilung, so dass Elemente einer Klasse
sich ähnlich verhalten

Basis für Klasseneinteilung

- Zuordnung nach Ressourcennutzung
- Zuordnung nach Anwendern
- Zuordnung nach Objekten die behandelt werden
- Zuordnung nach geographischen Gesichtspunkten
- Zuordnung nach funktionalen Gesichtpunkten
- Zuordnung nach Organisationseinheiten

Da analytische Modelle homogene Lasten benötigen,
in erster Linie Betrachtung der Ressourcennutzung

Charakterisierung des Benutzerverhaltens

zur Erinnerung:

Benutzerverhalten ist charakterisiert durch

- Benutzerverhaltensgraph (BVG) und
- Benutzerbesuchsmodell (BBM)

Aufbau der BVG Struktur durch Beobachtung und
Analyse des Benutzerverhaltens (siehe Kap. 2)

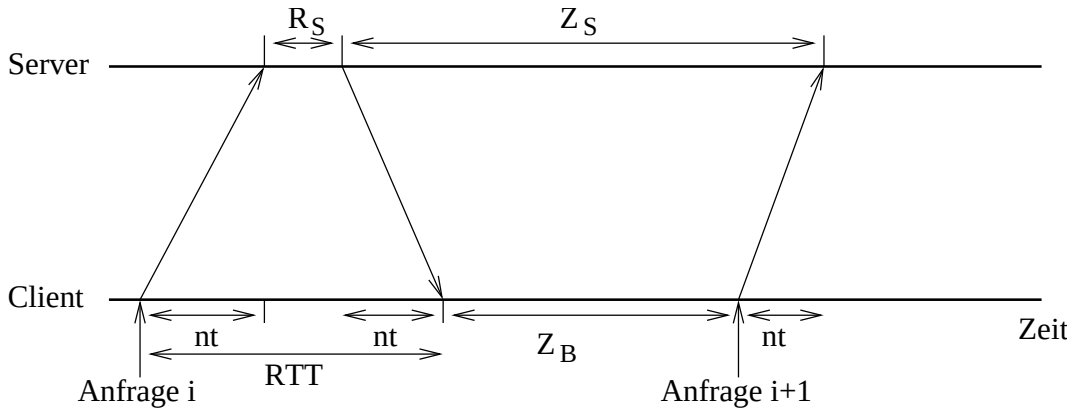
Hier nun Parametrisierung des BVGs auf Basis von
HTTP Logdateien

Logdateien werden von jedem Web-Server geschrieben

Größtes Problem Zuordnung von Aufrufen zu Benutzern und Sitzungen

- BVG beschreibt die Navigation von Kunden durch die Funktionen eines Web-basierten Systems
- BVG enthält zwei Arten von quantitativer Information
  - temporale Information über die Zeit, die zwischen dem Aufruf von zwei Funktionen vergeht
  - probabilistische Information über die Wahrscheinlichkeit bestimmte Sequenzen von Aufrufen durchzuführen

Temporale Aspekte aus Sicht des Servers beschrieben



nt = Übertragungszeit Z = Denkzeit am Browser

R_S= Antwortzeit Server Z_S = Denkzeit am Server

Charakterisierung des BVGs durch ein Paar von $n \times n$ Matrizen (P, Z) mit

- $P(i, j)$ enthält die Wahrscheinlichkeit, nach Ausführung von Funktion i, Funktion j auszuführen
- $Z(i, j)$ enthält die mittlere Denkzeit aus Server Sicht (Z_S) zwischen dem Aufruf von i und j

aus Kapitel 2 bekannt

$N = (I - P)^{-1}$ existiert und

$N(1, j) =$ mittlere Anzahl Aufrufe von Funktion j

($\Rightarrow$ BBM)

Wesentliche Quelle für Systemparameter HTTP-Logdateien

Information in der Logdatei hängt von der Konfiguration des Servers ab!

Übliches Format *Common Logfile Format*:

host Name oder IP-Adresse des Clients

ident falls freigegeben und vom Client unterstützt, Identitätsinformation

authuser falls benötigt (z.B. bei zugriffsgescgützten Dokumenten) Benutzername

date (Tag/Monat/Jahr:Stunde:Minute:Sekunde Zeitzone)

request Anfrage des Clients in "..."

status zurückgelieferter Statuscode, 3 Ziffern
z.B. 200 ok, 400 bad request, 503 service unavailable, ...

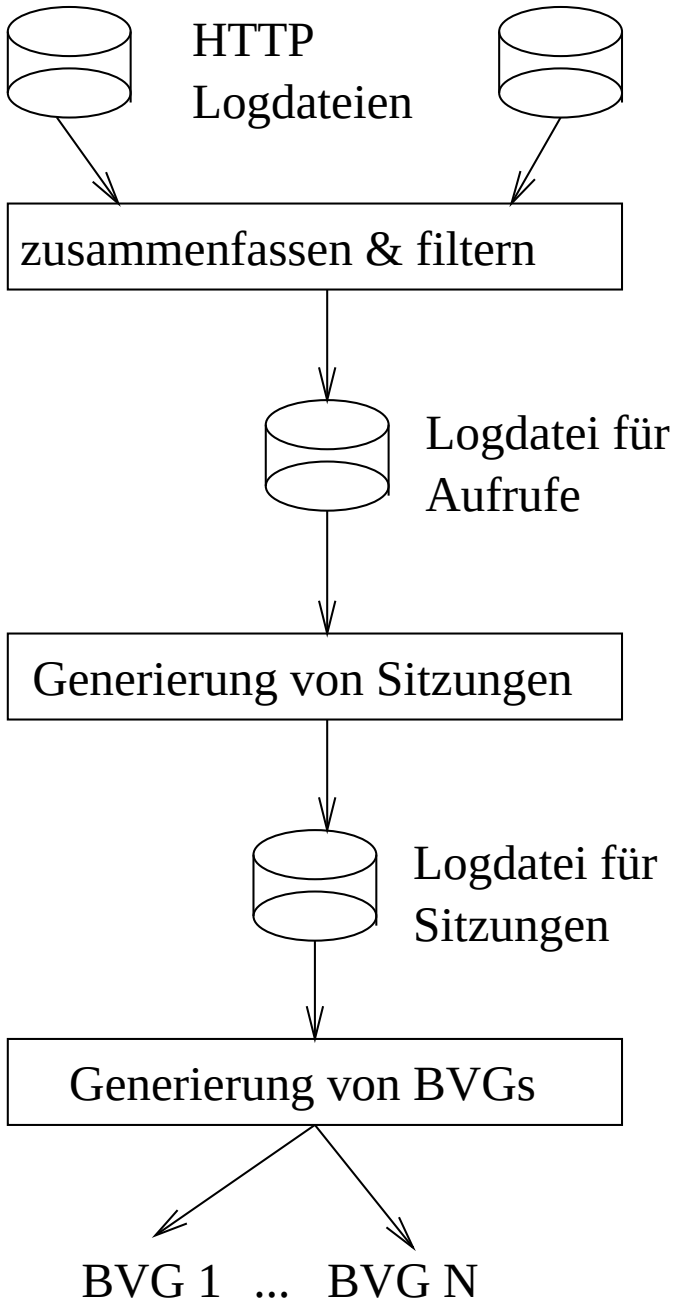
bytes Anzahl Bytes, die zum Benutzer zurückgeliefert wurden

Also standardmäßig keine Information über Ressourcenverbrauch beim Server

⇒properitäre Formate oder Nachbearbeitung der Logdatei

Parameterberechnung des BVGs aus HTTP Logdateien

Schritte der Parameterberechnung



Grundidee:

Detailliertes Verhalten zu wenigen Typen aggregieren!

Erster Schritt:

Zusammenfassung verschiedener Logdateien

(u.U. von mehreren Servern)

$\Rightarrow$ es entsteht Datei L mit Einträgen der Form (u, r, t, x) wobei

- u die Benutzerkennung des Clients ist, Gewinnung der Daten über Cookies, dynamische URLs oder Authentifzierungsdienste
- r der Type der Anfrage ist, z.B. GET Operation für eine Web Seite, Aufruf eines CGI Skripts
- t die Ankunftszeit der Anfrage beim Server ist
- x die Ausführungszeit der Anfrage ist

Zweiter Schritt:

Sortierung der Daten in L nach Benutzerkennung und Ankunftszeit,

Unterteilung der Sequenz für einen Benutzer in Sitzungen,

eine Sitzung ist beendet, wenn zwischen einer Anfrage und der Folgeanfrage vom selben Benutzer mehr als T Zeiteinheiten vergangen sind

(T typischerweise 30-60 Minuten)

Es entsteht Datei S mit einer Reihe von Sitzungen

Dritter Schritt:

Darstellung jeder Sitzung $(u_1, r_1, t_1, x_1), \ldots, (u_Q, r_Q, t_Q, x_Q)$
durch $n \times n$ Matrizen C und W,
die nach folgendem Algorithmus gebildet werden

$C(i,j) = 0$; für alle $i, j = 1, \ldots n$

$W(i,j) = 0$; für alle $i, j = 1, \ldots n$

for $k = 2$ to Q do

$C(r_{k-1}, r_k) = C(r_{k-1}, r_k) + 1;$

$W(r_{k-1}, r_k) = W(r_{k-1}, r_k) + (t_k - t_{k-1} - x_{k-1});$

end for ;

Vierter Schritt:

Bildung von K Klassen (gleich mehr zum Algorithmus)

Addition der Matrizen C und W aller Sitzungen

einer Klasse, s_k ist die Anzahl der Sitzungen in Klasse k

Fünfter Schritt:

Bildung eines BVGs pro Klasse k,

Berechnung der Matrizen P und Z mittels

$$P(i,j) = \frac{C(i,j)}{\sum_{l=1}^{n} C(i,l)} \text{ und } Z(i,j) = \frac{W(i,j)}{C(i,j)}$$

Die Ankunftsrate für Klasse k lautet $\lambda_k = s_k/T$,

wobei T die Länge des Beobachtungsintervalls ist

Zusammenfassung

- Aus den HTTP-Log-Dateien wird ein
  - quantifizierter Benutzerverhaltensgraph (BVG) und
  - ein Benutzerbesuchsmodell (BBM)
    unter Generierung von Benutzerklassen abgeleitet

- Für ein analytische Modell liegen damit folgende Parameter fest
  - Ankunftsrate für Anfragen jeder Klasse k als λ_k
  - Aufrufhäufigkeiten jeder Funktion i durch Klasse k

- Gebraucht werden zusätzlich noch
  - Bedienzeitanforderungen $D_{i,k}$ an den einzelnen Stationen

- Dazu müssen
  - weitere Messdaten erhoben werden
  - müssen Aktionen des BVG im CSID verfeinert werden

$\Rightarrow$ Methoden zur Klassenbildung und
detaillierten Messung

Parameterberechnung und Clusterbildung

Sei $w^j = (D_1^j, \ldots, D_K^j)$ die j-te beobachtete Komponente, wobei D_i^j der Bedienbedarf an Ressource i oder die Zeit zwischen dem $j-1$-ten und j-ten Aufruf von Funktion i ist

Seien p Beobachtungen vorhanden

Mittelwertbildung liefert

$$\overline{D_i} = \frac{1}{p} \cdot \sum_{j=1}^{p} D_i^j$$

als Bedienbedarf bzw. Zwischenankunftszeit

$$Var_i = \frac{1}{p-1} \cdot \sum_{j=1}^{p} (D_i^j - \overline{D}_i)^2$$

ist die Varianz des Bedienbedarfs/der Zwischenankunftszeit

Ziel niedrige Varianz innerhalb einer Klasse

Zum Vergleich besser geeignet Variationskoeffizient

$$VK_i = \frac{\sqrt{Var_i}}{\overline{D_i}}$$

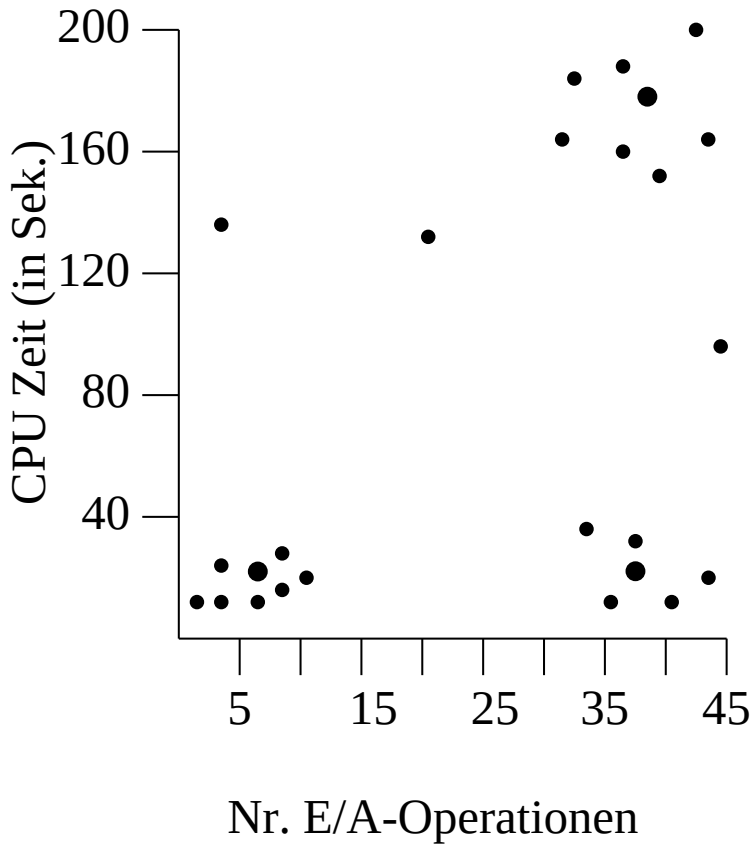
da "bereinigt" um Mittelwert

Varianz oder Variationskoeffizient erlaubt oft Entscheidung, ob Repräsentation der Last als eine Klasse adäquat

Falls nicht adäquat, Unterteilung in Klassen

auf Basis der Cluster-Analyse

Beispiel einer Last mit Belegung von zwei Ressourcen



Darstellung zeigt 3 Cluster und einige Ausreißer,
die in keinem Cluster liegen

Cluster-Analyse:

Algorithmus zur Bildung der Cluster auf Basis
gemessener Daten durch Minimierung einer Bewertungsfunktion

Cluster Analyse-Algorithmen in fast allen Statistikpaketen
vorhanden

Wie betrachten kurz das grundsätzliche Vorgehen

Reduktion des Datenbestands

Da Cluster Analyse aufwändig ist, wird bei einer großen Zahl von Beobachtungen in der Regel nur eine zufällige Auswahl zur Clusterbildung verwendet

oder

Daten schon aggregieren durch Zusammenfassung und Mittelwertbildung

Entfernen von Ausreißern

- Einzelne Ausreißer können Ergebnisse stark verfälschen und sollten nicht betrachtet werden
- Z.B. löschen der 1% kleinsten und 1% größten Werte
- Aber Vorsicht: In manchen Fällen beeinflussen selten auftretende Ereignisse mit einem großen Bedienbedarf das Systemverhalten (z.B. Job zur Reorganisation einer DB).
  Diese Werte nicht löschen

$\Rightarrow$ Löschen von Ausreißern erfordert Kenntnis des Systemverhaltens!

Skalierung der Daten

Zur besseren Handhabung und

um konsistente Ergebnisse zu erhalten (siehe unten)

werden Daten skaliert

Transformation in das Intervall $[0, 1]$

$$\widetilde{D}_i^j = \frac{D_i^j - \min_l(D_i^l)}{\max_l(D_i^l) - \min_l(D_i^l)}$$

Skalierung auf Mittelwert 0 und Varianz 1

$$\widetilde{D}_i^j = \frac{D_i^j - \overline{D}_i}{Var_i}$$

Abstandsmessung

Quantifizierung des Abstands zweier Messwerte

Jeder Messwert ist K-dimensionaler Vektor

Übliches Abstandsmaß für Vektoren

Euklidische Norm

Abstand d zwischen $w^j = (D_1^j, \ldots, D_K^j)$

und $w^l = (D_1^l, \ldots, D_K^l)$:

$$d = \sqrt{\sum_{k=1}^{K} (D_k^j - D_k^l)^2}$$

Euklidische Norm ist nicht unabhängig von

der Skalierung der Daten

Beispiel

Transaktion	CPU Zeit	Größe in Byte
Tr1	90	13000
Tr2	500	4000
Tr3	700	25000
Mittelwert	430	14000
Standardabweicheung	310.96	10535.65

3 Transaktionsklassen eines DB Servers

Euklidische Abstände:

$$d_{1,2} = \sqrt{(90-500)^2 + (13000-4000)^2} = 9009.3$$

$$d_{1,3} = \sqrt{(90-700)^2 + (13000-25000)^2} = 12015.5$$

$$d_{2,3} = \sqrt{(500-700)^2 + (4000-25000)^2} = 21001.0$$

$\Rightarrow$ Tr2 liegt näher an Tr1 als an Tr3

Falls Größe in KByte gemessen wird

$$d_{1,2} = \sqrt{(90-500)^2 + (12.7-3.9)^2} = 410.1$$

$$d_{1,3} = \sqrt{(90-700)^2 + (12.7-24.4)^2} = 610.1$$

$$d_{2,3} = \sqrt{(500-700)^2 + (3.9-24.4)^2} = 201.1$$

$\Rightarrow$ Tr2 liegt näher an Tr3 als an Tr1

$\Rightarrow$ Skalierung der Werte ist notwendig

Cluster Algorithmen

Es gibt zahlreiche Cluster-Algorithmen

Ziel aller Algorithmen:

Fasse Daten so zusammen, dass

- Elemente eines Clusters so ähnlich wie möglich
- Elemente unterschiedlicher Cluster so unterschiedlich wie möglich

Zentrum von Clusters j $(\overline{D}_1^j, \ldots, \overline{D}_K^j)$ ist durch
den Mittelwert aller Elemente im Cluster definiert

Man unterscheidet

- Hierarchische Techniken:
  Konsekutive Schritte fassen immer mehr Daten zusammen bis die gewünschte Zahl an Klassen erreicht ist
- Nicht-hierarchische Techniken:
  Ausgehend von einer Partitionierung in die gewünschte Klassenzahl werden Daten verschoben, um ein Optimalitätskriterium zu erreichen

Wir betrachten je einen populären Algorithmus pro Klasse

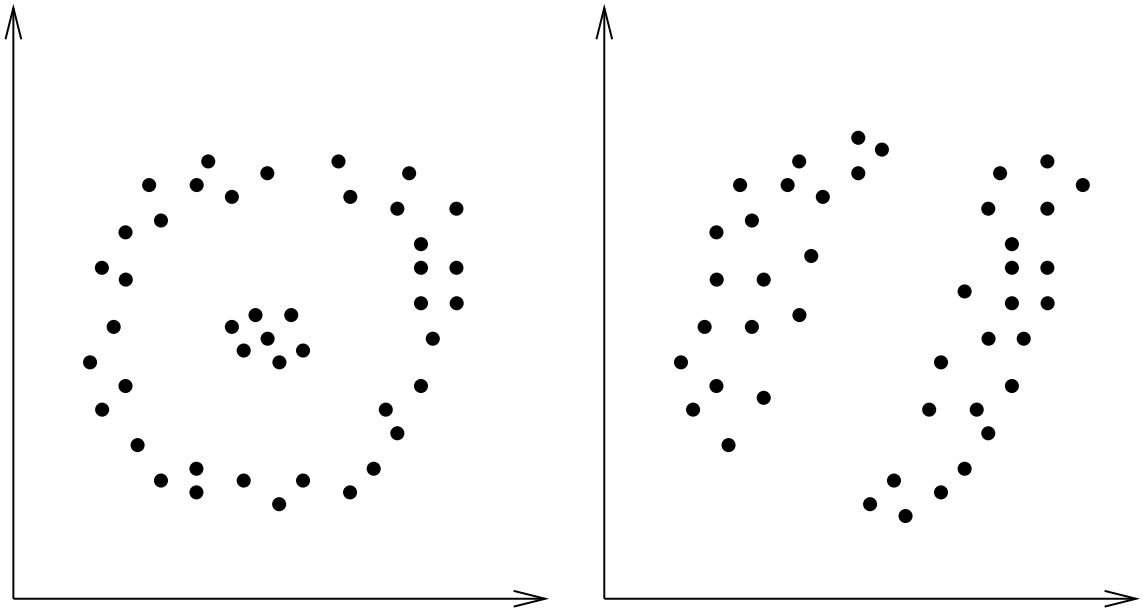
Minimale Spannbaumalgorithmus (hierarchisch)

1. Definiere p Cluster, die aus jeweils einem Element bestehen
2. Bestimme Zentren $\overline{D}_i^j$ für alle Cluster j, für die diese Werte noch nicht berechnet
3. Berechne für alle Clusterpaare j, l die Euklidische Distanz $d_{j,l}$ der jeweiligen Mittelwerte
4. Bestimme $\min_{j,l}(d_{j,l})$ und fasse Cluster j und l zu einem neuen Cluster zusammen
5. Falls die Anzahl der Cluster wie gewünscht, dann stop, sonst fahre bei 2. fort

Algorithmus des p-fachen Mittelwerts

1. Wähle p Punkte, die als Zentren der Cluster dienen (z.B. durch Auswahl der ersten p Elemente)
2. Ordne jedes Element dem Cluster zu, dessen Zentrum die kleinste Euklidische Distanz vom Element hat, berechne bei jeder Zuordnung das Zentrum neu
3. Führe den vorherigen Schritt so lange durch bis sich die Zuordnung der Elemente zu Clustern nicht mehr ändert

Grenzen der Cluster-Analyse



Clustering besser als zufällige Zusammenfassung aber

Resultate sind sehr variabel in Abhängigkeit

- vom Algorithmus
- vom Distanzmaß
- von der Skalierung

Verwendung von Clusteralgorithmen nicht "blind",
sondern in Kombination mit Interpretation
der gemessenen Daten

4.2 Benchmarks

Bester Weg zur Bestimmung der Leistung
eines vorhandenen Systems

Messung unter realer Last

Probleme:

- Was ist reale Last?
- Wie können unterschiedliche Systeme auf Basis realer Lasten verglichen werden?

Alternative:

Vergleich unterschiedlicher Systeme
auf Basis von Benchmarks

Benchmarks sind synthetische Lasten,
die "typisch" für gewisse
Anwendungsszenarien sind

Vorteile von Benchmarks:

- Zahlreiche Standardbenchmarks sind verfügbar
- Standardbenchmarks beschreiben typische Anwendungsszenarien
- Standardbenchmarks liefern einfache Vergleichswerte für unterschiedliche Systeme

Aber vorsicht:

Benchmark können den Benutzer informieren,

aber auch in die Irre führen!

Interpretation der Resultate ist wichtig

Zuerst folgende Fragen beantworten:

- Was testet der verwendete Benchmark?
- Wie gut repräsentiert der Benchmark die tatsächliche Last?
- Welche Größen misst der Benchmark?

Falls Benchmark richtig interpretiert wird, können mit Hilfe der gemessenen Resultate

- Systeme verglichen werden
- Parameter für Modelle bestimmt werden

Aber

Benchmarks sind gut geeignet

zum Vergleich von Systemen, aber

i.a. nicht geeignet zur Kapazitätsplanung

$\Rightarrow$ Benchmarks hier im wesentlichen zur

Parametrisierung von Modellen

4.2.1 Struktur von Benchmarks

Wesentlich Leistungsmaße:

- Für Benutzer: Antwortzeit, Kosten
- Für Betreiber: Durchsatz, Auslastung, Kosten

Vergleich von Systemen auf Basis von technischen Daten
ist i.a. nicht möglich

Taktrate des Prozessors oder Leistung in MIPS sagen
wenig über Systemleistung aus

Leistung wird bestimmt durch Befehlssatz
(Unterschied CISC-RISC), Speicherzugriffszeiten,
Cachegröße und -protokoll etc.

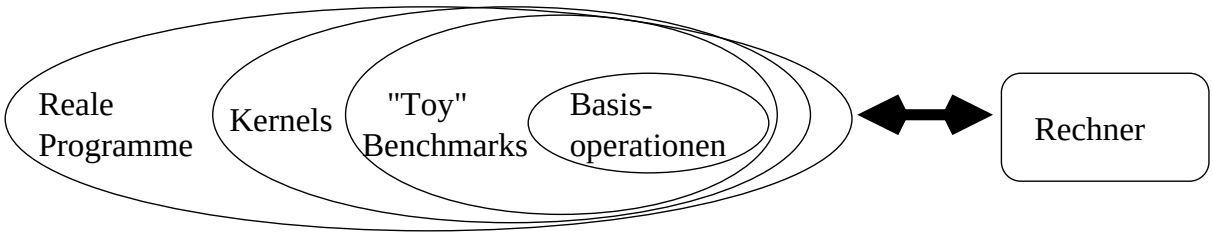
Systemleistung entsteht aus komplexem Zusammenspiel
von Hardware, Systemsoftware und Anwendungssoftware

In verteilten Systemen noch zusätzliche Aspekte
wie Kommunikationsnetz, Kommunikationsprotokolle
und Lastverteilung

Deshalb Leistungsvergleich auf Basis von
Programmausführungszeiten

Also Messung der Laufzeit eines Programms auf
unterschiedlichen Rechnern

Benchmarkhierarchie



- Benchmarks für Basisoperationen:
  - Untersuchung der Geschwindigkeit von Operationen wie Addition oder Multiplikation
  - Beispiele: Dhrystone, Whetstone
- Toy Benchmarks:
  - Kleine Programme die klassische Probleme darstellen, i. a. nur geringe praktische Relevanz
  - Beispiele: Türme von Hanoi, Sieb des Eratosthenes
- Kernel Benchmarks:
  - Teile realer Programme, die besonders viel Zeit verbrauchen, i. a. numerische Anwendungen
  - Beispiele: Livermore Loops, Linpack
- Reale Programme als Benchmarks:
  - Komplexe Programme aus vorgegebenen Anwendungsgebieten zur Leistungsmessung. Komplexeste aber auch realistischste Form der Leistungsmessung
  - Beispiele: TPC, SPEC, AIM

Interpretation von Resultaten aus Benchmarks erfordert große Sorgfalt

Erforderliche Informationen:

- Genaue Spezifikation des benutzten Benchmarks einschließlich Version und der verwendeten Eingabeparameter
- Genaue Spezifikation des Systemumgebung
  - Hardware Prozessor, Cache, Speicher, E/A-Geräte, Netzkomponenten etc
  - Systemsoftware: Betriebssystem, Compiler + Parameterwerte bei Compilierung, Transaktionsmonitore etc.
- Kenntnis der Situation zur Laufzeit
  - exklusiv als einzige Last
  - während des "normalen" Betriebs

Übertragung von Resultaten aus Benchmarks auf eigene Systeme erfordert Beantwortung folgender Fragen

- Ist die Systemumgebung von der die Benchmark Resultate stammen ähnlich zu meiner Konfiguration?
- Ist die im Benchmark dargestellte Last repräsentativ für die Last meines Systems?

Anforderungen an einen guten Benchmark

- Relevanz: Aussagekräftige Resultate für eine spezifische Problemklasse
- Verständlichkeit: Möglichst einfache Programmstruktur mit nachvollziehbaren Abläufen
- Skalierbarkeit: Anwendbarkeit auf eine große Klasse von Systemen und Konfigurationen
- Akzeptanz: Resultate müssen möglichst unverfälscht sein und von Anwendern akzeptiert sein

Zwei große Gruppen veröffentlichen Benchmarks

- System Performance Evaluation Consortium (SPEC)
  Gruppe führender Hersteller und Anwender, die
  ▷ standardisiert Benchmarks für zahlreiche Anwendungsgebiete
  ▷ Laufzeitresultate der Benchmarks auf unterschiedlichen Systemen
  veröffentlichen
- Transaction Processing Performance Council (TPC)
  nicht kommerzielle Organisation, die Benchmarks für Datenbankanwendungen veröffentlichen
  TPC-C und TPC-W sind weit genutzte Beispiele

4.2.2 Benchmarks auf Komponentenebene

Ziel der Benchmarks:

Messung der Leistung einzelner Komponenten

Wir betrachten als Beispiel die CPU Leistung

(andere Möglichkeiten wären E/A-Durchsatz,

Durchsatz Kommunikationssystem)

SPEC CPU Benchmarks dienen zum Vergleich

der CPU Leistung auf Basis

rechenintensiver Programme

Es gibt zwei "Suiten" von CPU Benchmarks in SPEC2000

- CFP für Fließkomma Operationen
  - 14 Programme in FORTRAN oder C geschrieben
- CINT für Integer Operationen
  - 12 Programme in C oder C++ für Integer Operationen

Leistung der Programme wird für eine

Referenzkonfiguration angegeben

(hier 300MHz Sun Ultra5_10 mit Faktor 100)

Angabe anderer Systemleistungen relativ

zur Referenzkonfiguration

SPEC CINT

Nr.	Benchmark	Anwendungsgebiet
1	164.gzip	compression
2	175.vpr	FGPA circuit placement
3	176.gcc	C-Compiler
4	181.mcf	combinatorical optimization
5	186.crafty	Chess
6	197.parser	Word processing
7	252.eon	Visualization
8	253.perlbmk	Perl programming
9	254.gap	Group theory
10	255.vortex	Object-oriented database
11	256.bzip	Compression
12	300.twolf	Place and route simulator

Für jeden Benchmark:

SPECratio nnn.benchmark $= \frac{nnn.benchmark\ Referenzzeit}{nnn.benchmark\ Laufzeit} \cdot 100$

SPECrate misst den Systemdurchsatz als Funktion der

Anzahl parallel laufender Kopien eines Benchmarks

Aus SPECint abgeleitete Resultate

- SPECint geometrischer (!) Mittelwert der SPECratio aller 8 Programme mit Optimierungen durch den Compiler

- SPECint_base geometrischer (!) Mittelwert der SPECratio aller 8 Programme ohne Optimierungen durch den Compiler
- SPECint_rate geometrischer (!) Mittelwert des maximal erreichbaren Durchsatzes bei beliebig vielen Kopien mit Optimierungen durch den Compiler
- SPECint_rate_base geometrischer (!) Mittelwert des maximal erreichbaren Durchsatzes bei beliebig vielen Kopien ohne Optimierungen durch den Compiler

Anwendungsbeispiel für Kapazitätsplanung:

Lohnt sich die Anschaffung eines neuen Web Servers?

Bisheriger Web Server erreicht SPECint = 363

Neues Modell würde erreichen SPECint = 489

Analyse auf des Systems auf Basis eines
Modells wie auf Folie 3.75/76 gezeigt

bisherige CPU Bedienzeit D_{CPU}^{alt}

neue CPU Bedienzeit $D_{CPU}^{neu} = D_{CPU}^{alt} \cdot \frac{363}{489} = \frac{D_{CPU}^{alt}}{1.35}$

Analyse des modifizierten Modells liefert Aussagen über
die Auswirkungen des Ersetzung des Servers

Leistungssteigerung in Relation zu Kosten setzen

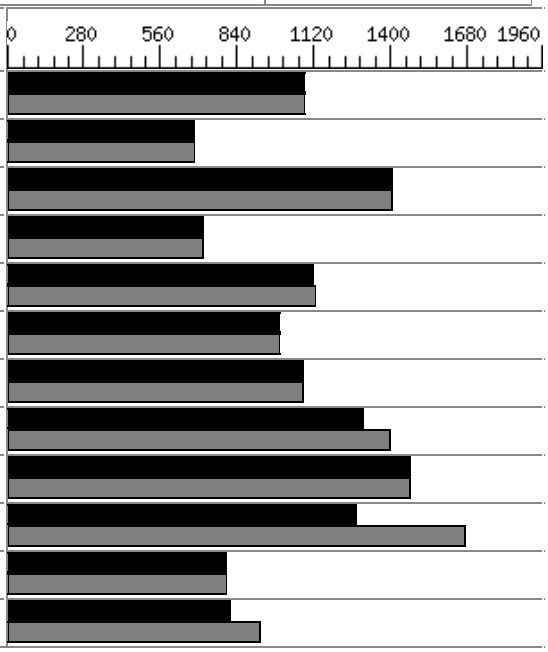
CINT2000 Result

Sun Microsystems
Sun Fire V65x (3.06 GHz Xeon)

SPECint2000 = 1066
SPECint_base2000 = 1024

SPEC license # 6	Tested by: Sun Microsystems	Test date: May-2003	Hardware Avail: May-2003	Software Avail: Apr-2003

Benchmark	Reference Time	Base Runtime	Base Ratio	Runtime	Ratio
164.gzip	1400	129	1082	129	1082
175.vpr	1400	206	679	206	679
176.gcc	1100	78.6	1399	78.6	1399
181.mcf	1800	253	711	253	711
186.crafty	1000	90.0	1111	89.1	1123
197.parser	1800	182	987	182	987
252.eon	1300	121	1072	121	1072
253.perlbmk	1800	139	1296	129	1394
254.gap	1100	75.1	1464	75.1	1464
255.vortex	1900	150	1269	114	1666
256.bzip2	1500	188	797	188	797
300.twolf	3000	373	804	327	917
SPECint_base2000			1024		
		SPECint2000			1066



Hardware		Software	
Hardware Vendor:	Sun Microsystems	**Operating System:**	Red Hat Linux 7.3
Model Name:	Sun Fire V65x (3.06 GHz Xeon)	**Compiler:**	Intel(R) C++ Compiler for 32-bit applications Version 7.1 Build 20030307Z C benchmarks compiled with icc C++ benchmarks compiled with icpc Microquill SmartHeap 6.02
CPU:	Intel Xeon (533MHz system bus)	**File System:**	ext2
CPU MHz:	3066	**System State:**	Multi-User (run level 3), hyperthreading disabled in BIOS
FPU:	Integrated		
CPU(s) enabled:	1 core, 1 chip, 1 core/chip (Hyper-Threading Technology disabled)		
CPU(s) orderable:	1,2		
Parallel:	No		
Primary Cache:	12K(I) micro-ops + 8KB(D) on chip		
Secondary Cache:	512KB(I+D) on chip		
L3 Cache:	None		
Other Cache:	None		
Memory:	2GB (2 x 1GB)		
Disk Subsystem:	1 x 36GB Seagate Ultra320 SCSI		
Other Hardware:	None		

Notes / Tuning Information

Datei Server

Test von Datei Server mit synthetischen Programmen,
die Zugriffe auf Dateien enthalten

SPEC System File Server (SFS) ist ein Benchmark zur
Analyse von File Servern in Unix Umgebungen

SFS basiert auf dem LADDIS Benchmark

- Programm zum Test von NFS
- Ausführung von verschiedenen NFS Kommandos
- 17 unterschiedliche Operationen
- Verteilung der Kommandos: 27% LOOKUP, 18% READ, 11% GETATTR, 9% READDIRPLUS, 9% WRITE, 7% READLINK, 7% ACCESS, 5% COMMIT, ...
- Programm parametrisierbar und an Umgebung anpassbar
- Anfragen an den Datei-Server werden als
kontinuierlicher Strom erzeugt (dies ist fragwürdig)

Durch Variation der Anzahl NFS Operationen
erhält man unterschiedliche Antwortzeiten

SPECnfs beschreibt die Anzahl der NFS Operationen pro
Sekunde, so dass die mittlere Antwortzeit 50 msek ist

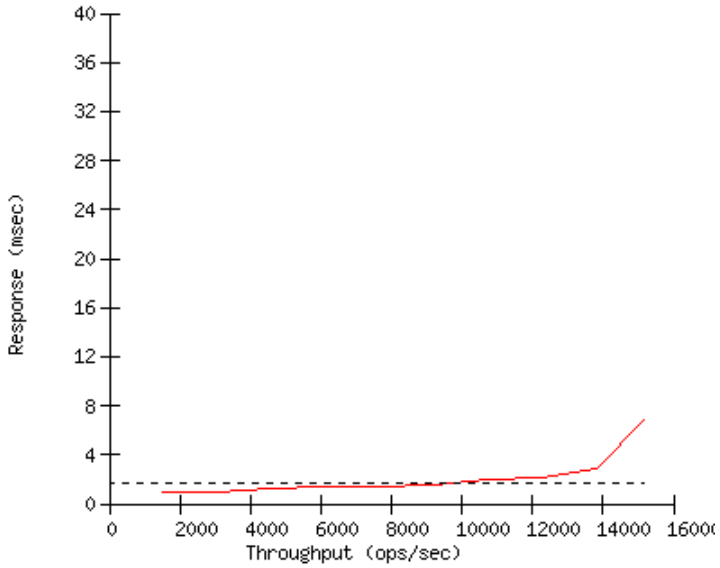
$\Rightarrow$ Ein Wert zum Vergleich von Systemen

SPECsfs97_R1.v3 Result

Sun Microsystems, Inc. : Sun Fire V250
SPECsfs97_R1.v3 = 15161 Ops/Sec (Overall Response Time = 1.74 msec)

Performance

Throughput (ops/sec)	Response (msec)
1514	0.9
3064	1.0
4560	1.2
6113	1.4
7636	1.4
9191	1.6
10754	2.0
12268	2.1
13834	2.8
15161	6.8



Configuration

Server Configuration and Availability	
Vendor	Sun Microsystems, Inc.
Hardware Available	October 2003
Software Available	December 2003
Date Tested	December 2003
SFS License Number	6
Licensee Locations	Santa Clara, CA

Network Subsystem	
Network Type	on-board 10/100/1000-Mbps BaseT
Network Controller Desc.	Gigabit Ethernet
Number Networks	1 (N1)
Number Network Controllers	1
Protocol Type	TCP

CPU, Memory and Power	
Model Name	Sun Fire V250
Processor	1.28GHz UltraSPARC IIIi
# of Processors	2
Primary Cache	32KBI + 64KBD on chip

Switch Type	Extreme / Summit 7i (fibre) and Asante GX5-2400 (copper)
Bridge Type	N/A
Hub Type	N/A
Other Network Hardware	N/A

Secondary Cache	1MB on chip
Other Cache	N/A
UPS	N/A
Other Hardware	N/A
Memory Size	8 GB
NVRAM Size	8 GB (1 GB per Sun StorEdge 6020 RAID Controller)
NVRAM Type	Sun StorEdge 6020 write cache
NVRAM Description	auto-sensing WB/WT depending on battery charge

Server Software	
OS Name and Version	Solaris 9 12/03
Other Software	N/A
File System	UFS
NFS version	3

Server Tuning	
Buffer Cache Size	dynamic
# NFS Processes	dynamic, up to a maximum of 1024
Fileset Size	146.0 GB

Disk Subsystem and Filesystems	
Number Disk Controllers	3 (1 on-board Ultra 160 SCSI, 2 PCI Dual FC Adptrs)
Number of Disks	113
Number of Filesystems	16 (F1-F16)
File System Creation Ops	default
File System Config	(2) 230GB file systems per 14-disk RAID0 LUN

Disk Controller	On-board Ultra160 SCSI
# of Controller Type	1
Number of Disks	1
Disk Type	Seagate Cheetah 36GB 10K rpm (ST-336607LW)
File Systems on Disks	OS, swap
Special Config Notes	N/A

Disk Controller	Sun StorEdge 2Gb PCI Dual FC Network Adapter
# of Controller Type	2
Number of Disks	112
Disk Type	Seagate Cheetah 36GB 15K rpm (ST-336753FC)
File Systems on Disks	F1-8,F9-16
Special Config Notes	4 file systems on 2 RAID 0 LUNs per Sun StorEdge 6120HA Array (see Other Notes)

Load Generator (LG) Configuration	
Number of Load Generators	3
Number of Processes per LG	32
Biod Max Read Setting	2
Biod Max Write Setting	2

LG Type	LG1
LG Model	Sun Enterprise 420R
Number and Type Processors	4x450MHz UltraSPARC II
Memory Size	4GB

Operating System	Solaris 9 8/03
Compiler	Forte Developer 7 C 5.4 2002/03/09
Compiler Options	-xarch=v9 -DPORTMAP -lm -lsocket -lnsl
Network Type	Sun GigabitEthernet PCI adapter 2.0

Testbed Configuration				
LG #	LG Type	Network	Target File Systems	Notes
1,2,3	LG1	N1	/F1-16, /F1-16	N/A

Other Notes

- Tuning:
- rlim_fd_max=70000, rlim_fd_cur=70000, autoup=600, tune_t_fsflushr=10
- maxphys=1048576, maxusers=2048, lotsfree=1024, tcp_conn_hash_size=262144
- bufhwm=1048576, ncsize=2097152, ufs_ninode=1048576, segmap_percent=70
- ufs_HW=4194304, ufs_LW=1048576
-
- There were 2 Sun StorEdge 2Gb PCI Dual FC Network Adapters in the
- tested configuration. Each connects to 2 Sun FCAL 7-port hubs.
- Both channels of a Sun StorEdge 6120HA array are connected to each
- hub, for a total of 4 Sun StorEdge 6120HA arrays in the tested
- configuration, 2 Sun StorEdge 6120HA arrays per Sun StorEdge 2Gb Dual
- FC Network Adapter.
-
- Each Sun StorEdge 6120HA array consists of 2 6020 subarrays of 14
- disks each, configured as one RAID 0 LUN striped across 14 disks. Two
- file systems were configured on each RAID 0 LUN.
-
- Sun StorEdge 6120HA arrays include redundant batteries with
- sufficient capacity to allow an orderly shutdown of the array in
- the case of AC power loss. In case AC power is lost, the array
- will run from the batteries for a time sufficient to flush all data
- in its cache to disk, after which the array will shutdown.

4.2.3 Benchmarks auf Systemebene

Ziel: Aussagen über das Verhalten des gesamten Systems

$\Rightarrow$ Verwendung relativ komplexer Anwendungen
als Benchmarks

Wir betrachten Untersuchung von

- Datenbanken
- Web-Servern

Datenbanken

Große Datenbanken sind Teile einer Client/Server Infrastruktur

Benchmarks messen

- Prozessoren
- E/A-Subsysteme
- Betriebssystem(e)
- DB Management Systeme
- Transaktionsmonitore

Also Aussagen über das globale Verhalten

In unserem Kontext interessant:
Resultate der Benchmarks zur Bestimmung von Parameter für Modelle auf Systemebene (einzelne Stationen bzw. Zustands-/Transitionssysteme)

Oft verwendet Benchmarks TPC-C, -H, -R, -W

TPC-H und TPC-R

Emulation komplexer Datenbankabfragen

- TPC-H enthält gut strukturierte und vorab bekannte Anfragen
  (⇒ Möglichkeit der Anfrageoptimierung)
- TPC-R enthält ad hoc Anfragen
  (⇒ keine Möglichkeit der Anfrageoptimierung)

Datenbankschema besteht aus 8 Basistabellen, die für das zu messende System skaliert werden

Skalierungsfaktoren 1, 10, 30, 100, 300, 1000, 3000, 10000 wobei 100 einer DB-Größe von 100 Gbytes entspricht

Last der Benchmarks

- 22 Leseanfragen und 2 Lese-/Schreibanfragen
  (in SQL-92)
- Analyse im Ein- und Mehrbenutzerbetrieb

Basisleistungsgröße:

- QphH/QphR Anzahl abgearbeitete Anfragen pro Stunde + Skalierung des Wertes durch den gewählten Skalierungsfaktors
- Zusätzlich Angabe des Preis-/Leitungsverhältnises
  (in Anfragen pro Euro)

TPC-C

- Standardbenchmark für OLTPs
- Modelliert die Auftragsbearbeitung eines Großhandels
- Umfasst 5 Transaktionen:
  - Auftragseingang (45%)
  - Bezahlung (43%)
  - Auslieferung (4%)
  - Auftragsstatus (4%)
  - Bestandskontrolle (4%)

Transaktionen führen zum

- auslesen von Daten
- ändern von Daten
- löschen von Datensätzen
- einfügen von Datensätzen

Verteilung der einzelnen Transaktionen im Benchmark
ist repräsentativ für große Klasse von Anwendungen

Ergebnis von TPC-C wird in tpmC gemessen:
Maximale Anzahl von Auftragseingangstransaktionen,
die pro Minute bearbeitet werden, wobei für jeden
Transaktionstyp Antwortzeitschranke eingehalten wird
(z.B. 90% der Auftragseingänge in weniger als 5 Sek.)

Angabe der Ergebnisse

Konfiguration	Daten
Firma	X
Typ	xyz
Prozessoren	4
Plattenkapazität	2.61 Terabytes
DMBS	Micorsoft SQL
Betriebssystem	Windows NT
Transaktionsmonitor	Microsoft COM+
Kosten	445747 $
tpmC	34600
Preis/Leistung	12.89 $

Systemkonfiguration erlaubt

- 34600 Auftragseingangstransaktionen
- kosten 445747 $

Kosten umfassen Hardware, System- und DB-Software

Preis-/Leistungsverhältnis gibt den Preis pro tpmC an und
kann zum Vergleich unterschiedlicher System benutzt werden

Beobachtung:

Am oberen Ende des Leistungsspektrums
steigt Preis/Leistungsverhältnis

Web Server

Meist verwendet Webstone und SPECweb

Beide simulieren Browser,

die Anfragen an einen Web Server senden

Üblicherweise in LAN Umgebung zum Test

von Web Servern verwendet

Umgebung unterscheidet sich deutlich von

realer Web Server Umgebung (großes WAN)

Verwendung im WAN Kontext beeinflusst Messungen

durch schwer vorhersagbare Hintergrundlast

Webstone

- Konfigurierbarer Benchmark für HTTP Server
- Im wesentlichen GET Operationen für Web-Seiten,
- In neueren Versionen auch Ausführung von CGI Skripten
- Verteilter Mehrprozess-Benchmark
  - ▷ ein Master Prozess startet mehrere Client Prozesse,
  - ▷ die jeweils HTTP Anfragen generieren
  - ▷ bei Terminierung sammelt Master Prozess von den Clients gemessene Werte und generiert Resultate

Webstone enthält vier Klassen von Web Seiten,

die typische Seiten repräsentieren

Vom Benutzer spezifizierte Parameter

- Anzahl Prozesse
  (jeder Client generiert neue Anfrage sobald eine Anfrage vom Server beantwortet)

- Zugriffswahrscheinlichkeiten auf einzelne Seitentypen

- Anzahl der Seiten auf dem Server

- Anzahl der Rechner, auf denen Clients laufen

Resultate

- Durchsatz in Seiten pro Minute (Webstone number)

- Latenzzeit in Sekunden
  (Zeit vom Absenden bis zur vollständigen Übertragung)

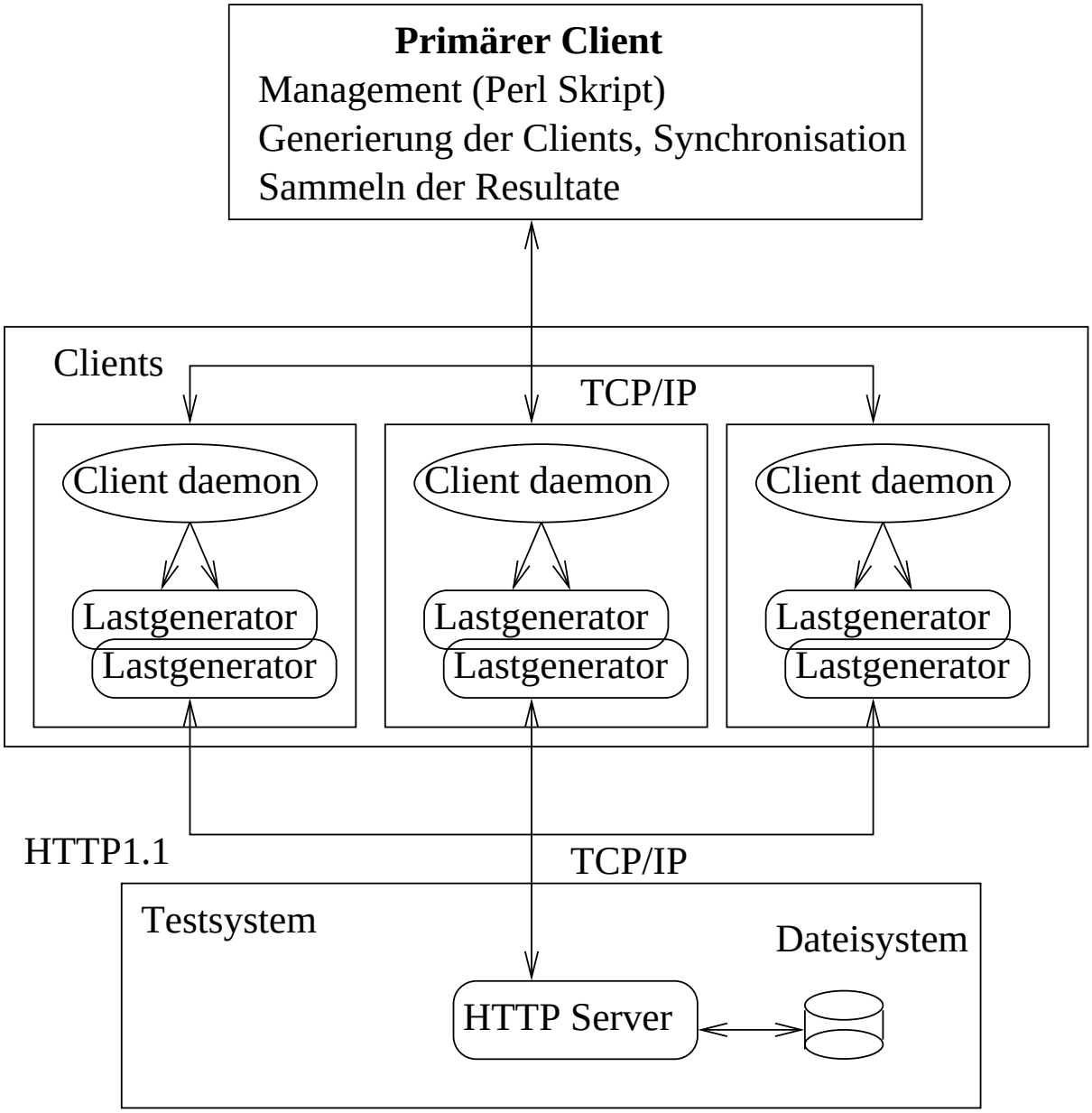
Ergebnisdarstellung (nach 10 minütiger Messung)

Metric	Value
Webstone number	456
Total number of clients	24
Total number of pages retrieved	4567
Total number of errors	0
Total number of connections	12099
Average time per connect in sek.	0.0039
Maximum time per connectin sek.	0.0370
Total number of bytes moved	3076611677
Average throughput (bytes/sek.)	215181
Average response time (msek.)	1181
Maximum response time (msek.)	18488

SPECweb

Leistungsmessung für Unix und Windows NT basierte Web Servern

Struktur des Benchmarks



Zahl der Clients hängt vom Testsystem ab

Laststruktur

keine Veränderungen der Last durch Benutzer möglich

- Messung simultaner Benutzerzugriffe
- PHP und JSP Implementierungen sind enthalten
- Bilder werden über 2 parallele HTTP-Verbindungen geladen
- Lastunterteilung in Bankanwendung (https), ecommerce (https und http) und normal (http)
- Caching werden nachgebildet

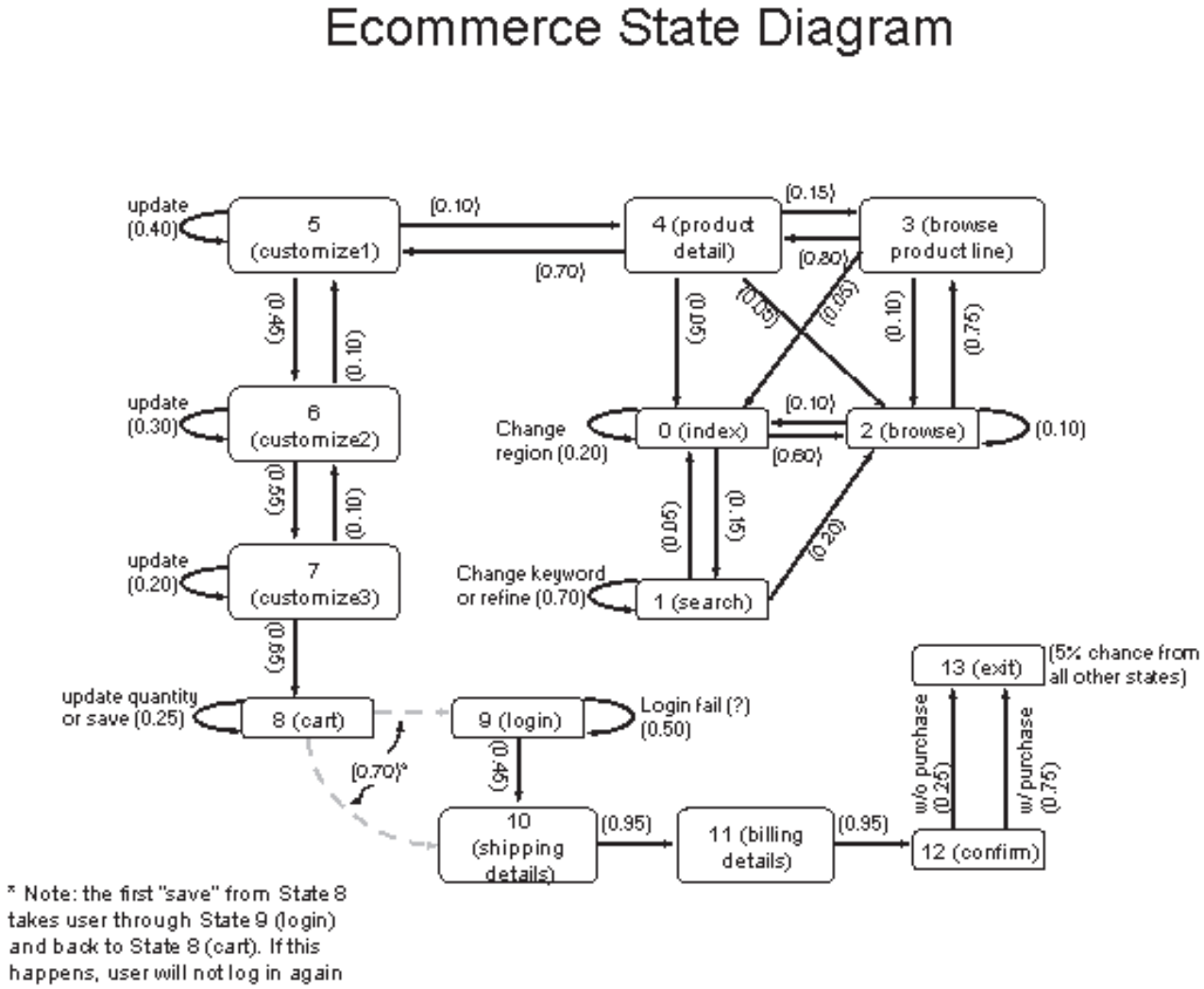
Resultate

$$SPECweb2005 = \sqrt[3]{\frac{bank\_measure}{bank\_reference} \cdot \frac{ecommerce\_measure}{ecommerce\_reference} \cdot \frac{normal\_measure}{normal\_reference}} \cdot 100$$

SPECweb2005 maximale Zahl simultaner Verbindungen unter Einhaltung vorgegebener Bedingungen

- Bank und Ecommerce: 95% der Antworten in weniger als 2 sek., 99% in weniger als 4 Sek.
- Normal: Jeder Benutzer versucht mit 100.000 Bytes/Sek. Seiten zu laden, 95% der Seiten müssen mit 99.000 Bytes/Sek. und 99% der Seiten müssen mit 95.000 Bytes/Sek. geladen werden

Beschreibung der Abläufe durch Markov-Ketten
(diese werden simuliert!)



SPECweb2005 Result

Sun Microsystems, Inc.: Sun Fire T2000 Sun Microsystems, Inc.: Sun Java[TM] System Web Server 6.1 SP5 64-bit	SPECweb2005 = 14001 SPECweb2005_Banking = 21500 SPECweb2005_Ecommerce = 21500 SPECweb2005_Support = 13160

Tested By: Sun Microsystems Inc.	SPEC License #: 6	Test Date: Nov-2005

Performance and Configuration Section	Details Section	Notes Section	Errors Section
Performance Section Configuration Section	Banking Details Ecommerce Details Support Details	Banking Notes Ecommerce Notes Support Notes	Banking Errors Ecommerce Errors Support Errors

Performance

Banking

Simultaneous User Sessions	Test Iteration	Aggregate QOS Compliance			Validation Errors
		Good	Tolerable	Fail	
21500	1	97.6%	99.0%	0.9%	0
	2	98.1%	99.3%	0.6%	0
	3	98.1%	99.2%	0.7%	0

Ecommerce

Simultaneous User Sessions	Test Iteration	Aggregate QOS Compliance			Validation Errors
		Good	Tolerable	Fail	
21500	1	98.4%	99.2%	0.7%	0
	2	99.1%	99.6%	0.3%	0
	3	98.5%	99.3%	0.6%	0

Support

Simultaneous User Sessions	Test Iteration	Aggregate QOS Compliance			Validation Errors
		Good	Tolerable	Fail	
13160	1	95.4%	99.0%	0.9%	0
	2	95.9%	99.4%	0.5%	0
	3	96.0%	99.4%	0.5%	0

Configuration

Availability Dates	
SUT Hardware	Nov-2005
Backend Simulator	Oct-2005
Web Server Software	Jan-2006
Operating System	Nov-2005
Other Components	Feb-2006

System Under Test (SUT)	
# of SUTs	1
Vendor	Sun Microsystems, Inc.
Model	Sun Fire T2000
Processor	Sun UltraSPARC T1
Processor Speed (MHz)	1200 MHz
# Processors	8 cores, 1 chip, 8 cores/chip (4 threads/core)
Primary Cache	16KB(I)+8KB(D) per core
Secondary Cache	3 MB per chip
Other Cache	N/A
Memory	32 GB
Disk Subsystem	3x73GB 10K RPM SAS
Disk Controllers	Onboard SAS Controller
Operating System	Solaris 10 3/05 HW2
File System	UFS
Other Hardware	1x Sun StorEdge 3510 RAID (dual raid, 12x73GB 15K RPM FCAL)+ 2x StorEdge 3510 JBOD (12x73GB 15K RPM FCAL)

Web Server Software	
Vendor	Sun Microsystems, Inc.
Name/Version	Sun Java[TM] System Web Server 6.1 SP5 64-bit
Dynamic Scripts	JSP
Server Cache	N/A
Log Mode	Common Log Format

Clients	
# of Clients	35
Model	Sun Fire V20z
Processor	AMD Opteron 248
Processor Speed (MHz)	2200 MHz
# Processors	2 cores, 2 chips, 1 core/chip
Memory	4 GB
Network Controller	1x Onboard Broadcom
Operating System	Solaris 10 3/05
JVM Version	Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_01-b08)
JIT Version	Java HotSpot(TM) Server VM (build 1.5.0_01-b08, mixed mode)
Other Hardware	N/A
Other Software	N/A

Backend Simulator (BESIM)	
# of Simulators	1
Model	Sun Fire V40z
Processor	AMD Opteron 880

TPC-W Benchmark zur Messung von E-Business Systemen

Basisszenario B2C:

- Web Server zum Verkauf von Diensten und Produkten über das Internet

Funktionen (Auswahl):

- Produktansicht
- Kundenregistrierung
- Auftragseingabe
- Auftragsstatus

Interaktionen, die vertrauliche Daten beinhalten, werden verschlüsselt

(SSL mit RSA, RC4 und MD5 Verschlüsselung)

Skalierbarkeit von TPC-W über die Kataloggröße von

1000 bis 10000000 Produkte

Unterscheidung in zwei Interaktionstypen

Browse: ansehen von Information ohne Auftragseingabe,

zugehörige Funktionen: Home, Browse, Select,

Product Detail, Search

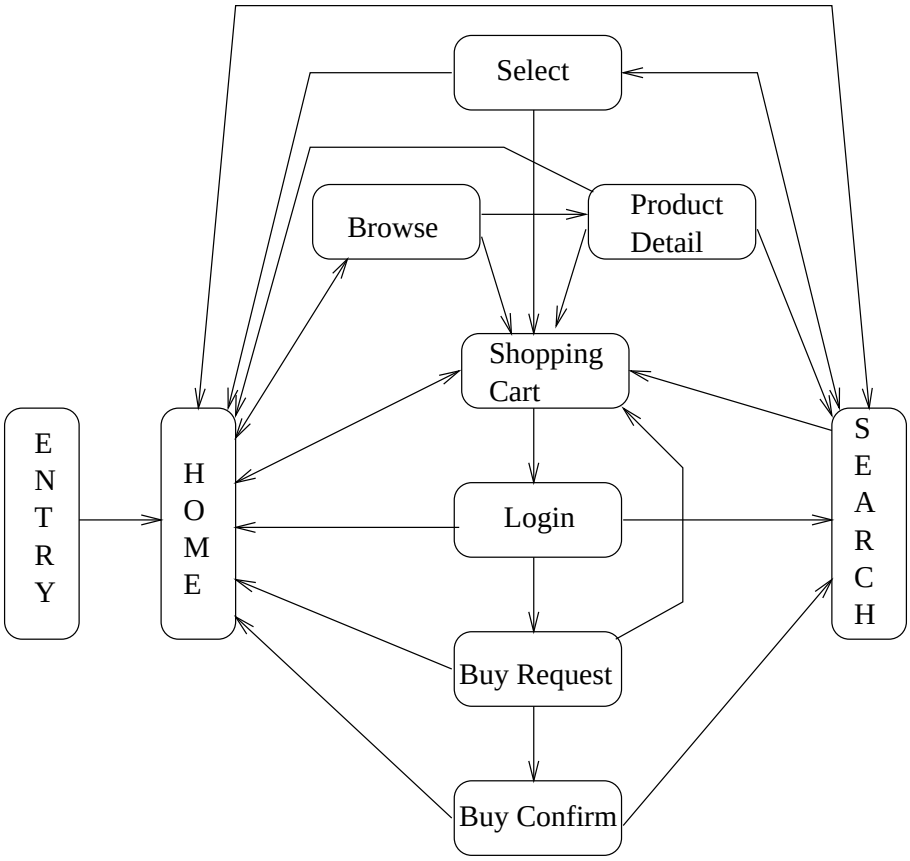
Order: Auftragseingabe und -abwicklung,

zugehörige Funktionen: Shopping Cart, Login,

Buy Request, Buy Confirm

Vereinfachter BVG von TPC-W

(ohne Transitionen zum Ausgang)



TPC-W spezifiziert drei Sitzungstypen

- Browsing Mix: 95% Browse Interaktionen und 5% Order Interaktionen (0.69% der Besucher kaufen)
- Shopping Mix: 80% Browse Interaktionen und 20% Order Interaktionen (1.2% der Besucher kaufen)
- Odering Mix: 50% Browse Interaktionen und 50% Order Interaktionen (10.18% der Besucher kaufen)

Leistungmaße von TPC-W (im stationären Zustand)

Länge des Messintervalls mind. 30 Minuten

Leistungsmaße unter vorgegeben Antwortzeit- und Fehlerschranken

- Durchsatzmaße
  - WIPS (Web Interactions Per Second) durchschnittliche Anzahl von Interaktionen des Shopping Mixes pro Sekunde
  - WIPSb durchschnittliche Anzahl von Interaktionen des Browsing Mixes pro Sekunde
  - WIPS0 durchschnittliche Anzahl von Interaktionen des Ordering Mixes pro Sekunde

- Kostenmaße
  - Metrik $ per Wips als Relation zwischen Systemkosten und Leistung

4.3 Messung

Verfügbarkeit von Daten ist notwendig zur

- Erstellung von Lastmodellen
- Parametrisierung von Systemmodellen

Daten können gewonnen werden aus

- Systemspezifikationen
- Veröffentlichungen
- Studien ähnlicher Systeme
- Kenntnissen über das System
- Benchmarks
- Messungen am System

Üblicherweise Messung mit Hilfe von Monitoren

Grundsätzliche Fragestellungen:

- An welchen Stellen soll gemessen werden?
- Welche Monitore sollen verwendet werden?
- Wie sollen die Modellparameter aus den gemessenen Daten gewonnen werden?

4.3.1 Vorgehen bei der Leistungsmessung

Messung kann interpretiert werden als Beobachtung des Systems über einen Zeitraum bezüglich der Variablen, die relevant sind für das quantitative Systemverhalten

Aufgabe des Systems Abarbeitung von
Anwendungsprogrammen unter Einhaltung der SLAs
$\Rightarrow$ Anwendungsorientierte Messung von Daten

Also Rückführung der gemessenen Daten
auf die verursachende Anwendung

Verteilte Systeme erfordern verteilte Messumgebungen

Schritte bei der Durchführung von Messungen

1. Referenzpunkte definieren
2. Messwerte definieren
3. Instrumentierung und Datenerhebung
4. Analyse und Transformation der Daten

Messtechniken:

- Event Mode: Daten werden zur Ereigniszeitpunkten gesammelt, z.B. bei Eintreten eines E/A-Interrupts
- Sampling Mode: Datensammlung zu vordefinierten Zeitpunkten, Aktivierung der Messroutine durch einen Timer

4.3.2 Monitore zur Datensammlung

Monitor ist ein Tool zur Beobachtung von Aktivitäten in einem System

Idealerweise sollte ein Monitor ein Beobachter des Systemverhaltens sein und das Systemverhalten in keiner Weise beeinflussen

Man unterscheidet

- Online Monitore: Systemzustand zur Laufzeit
- Batch Monitore: Datensammlung und spätere Analyse

Einige Kenngrößen von Monitoren

- Overhead: Belastung des Systems durch den Monitor
- Messbarkeitsbereich: Zustandsinformation, auf die der Monitor zugreifen kann
- Auflösung: Minimaler Zeitabstand zwischen zwei aufgezeichneten Ereignissen
- Eingangsrate: Mittlere und maximale Ereignisrate

Monitortypen

- Hardware Monitore
- Software Monitore
- Hybride Monitore

Hardware Monitore

Messumgebung auf Hardwareebene

Elektronische Signale und damit der Zustände von Hardwarekomponenten werden gemessen

Vorteile von Hardwaremonitoren

+ Als externe Hardware keine oder kaum Beeinflussung des Systemverhaltens

+ Unabhängig von der verwendeten Software

+ Hohe Eingangsrate und Auflösung

Nachteile von Hardwaremonitoren

- Hohe Kosten

- Steigende Systemintegration reduziert die möglichen Messpunkte zu wählen

- Keine Möglichkeit Messwerte Anwendungsprogrammen zuzuordnen

⇒ Letzter Punkt bedingt, dass Hardwaremonitore

- kaum zur Parametrisierung von Modellen geeignet und
- damit kaum zur Kapazitätsplanung einsetzbar sind

Softwaremonitore

Programm zur Sammlung von Statusinformationen und Ereignissen in einem System

Softwaremonitore

- können Informationen auf unterschiedlichen Ebenen sammeln
- können Ereignisse Anwendungsprogrammen zuordnen
- werden per Interrupt aktiviert
- können Daten direkt aufbereiten und aggregieren

Aber

- Softwaremonitore konkurrieren als Programme mit anderen Prozessen des zu messenden Systems ($\Rightarrow$ Interferenz, Overhead)
- sind vom verwendeten Betriebssystem abhängig und damit nur eingeschränkt portabel

Hybride Monitore

Kombination aus Hardware- und Softwaremonitor

$\Rightarrow$ Hohe Auflösung und geringer Overhead des Hardwaremonitors und Flexibilität und Datenaufarbeitungsmöglichkeiten des Softwaremonitors

Messwerkzeuge für Netze

Man unterscheidet

- passive Werkzeuge, die nur das Netzverhalten beobachten
  - netramet: Messung auf Routern
  - tcpdump und tcptrace: Ausgabe der Header von Paketen am Interface und deren Auswertung

- aktive Werkzeuge, die Pakete erzeugen und messen
  - ping: Messung von RTT
  - timeit: Zugriffszeiten auf Web-Seiten

Werkzeuge zur Messung von Servern

- Windows NT Performance Monitor:
  - Standardwerkzeug in NT zur Leistungsbeobachtung
  - messbare Größen oft nicht ausreichen zur Parametrisierung analytischer Modelle

- Unix sar:
  - Ausgabe systemweiter Statistiken in Unix Systemen

4.3.3 Parameterbestimmung

Ziel: Parametrisierung des Last- und Systemmodells

$\Rightarrow$ Daten müssen sich auf die im Modell beschriebenen Ressourcen beziehen

Aufteilung der Last in Klassen

- bzgl. Anwendungen mit unterschiedlichem Ressourcenbedarf (Cluster Analyse)
- bzgl. Anwendungen, die separat behandelt werden sollen (z.B. separate Analyse oder Prognose)
- bzgl. Entscheidung ob Klassen offen oder geschlossen

Bestimmung der Lastintensitäten

- Population und Denkzeit im geschlossenen Modell
- Ankunftsrate im offenen Modell

Bestimmung der Bedienzeiten

u.U. indirekte Bestimmung über Satz von Little, wenn direkte Messung nicht möglich

z.B. System wir für T Zeiteinheiten beobachtet, Auslastung während dieser Zeit war U_i und C_0 Aufträge wurden bedient $\Rightarrow$ Bedienzeit $D_i = U_i \cdot T / C_0$

Oft sind nur Teile der Ressourcenbelastung messbar und
im Modell berücksichtigt

Typisches Problem BS Overhead im Modell

- konstanter Anteil unabhängig von der Last
- lastabhängiger Anteil

Zwei Möglichkeiten der Berücksichtigung im Modell

- Overhead als separate Lastklasse
- Overhead als zusätzliche Zeit bei jeder Bedienung

Messung, insbesondere in verteilten Systemen,
liefert oft keine konstanten Werte
d.h. mehrere Messungen des selben Ablaufs
führen zu unterschiedlichen Ergebnissen

- Mehrere Messungen durchführen
- Mittelwerte bestimmen
- Konfidenzintervalle bestimmen
- U.U. Sensitivitätsanalyse des Modells bzgl. der gemessenen Parameter

4.4 Lastvorhersage

Ziel der Kapazitätsplanung: Aussagen über die Zukunft

Dies erfordert Vorhersagen über die zukünftige Last,
die ein System verkraften muss

Vorhersagen sind Grundlage für Planungsentscheidungen,
die einen gewissen Vorlauf brauchen

Schlechte Lastvorhersagen führen zu schlechten Planungen

- Fehlender Kapazität mit Umsatzverlusten
- Zuviel Kapazität mit zu hohen Kosten

Auch bei guten Modellen gilt:

Zukünftige Last kann größer,
aber auch kleiner als die aktuelle Last sein

Gründe für Laständerungen

- saisonale Gründe
- neue Anwendungen
- neue Technologien
- neue Benutzer

Lastvorhersage erfordert ein systematisches Vorgehen
unter Verwendung unterschiedlicher Prognosemethoden

4.4.1 Eine Strategie zur Lastvorhersage

Vorhersagemethoden kann man unterteilen in

- qualitative Methoden
  subjektive Formulierung von Erwartungen über die Zukunft auf Basis von
  - Intuition
  - Expertenmeinungen
  - historischen Analogien
  - technischen Entwicklungen
- quantitative Methoden
  - Schätzung zukünftiger Daten aus historischen Daten
  - Beschreibung des Datenverlaufs durch ein mathematisches Modell

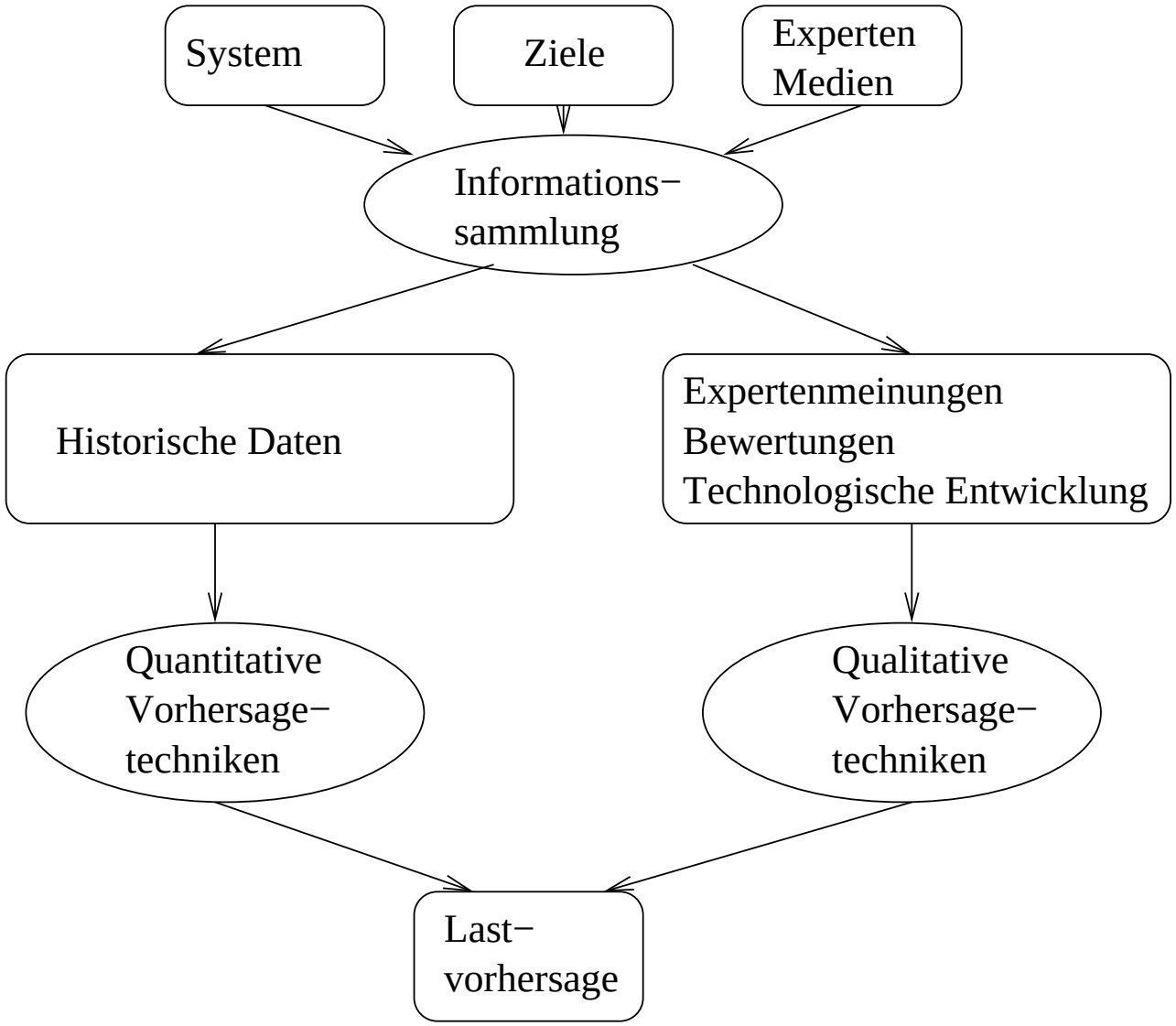
Probleme treten immer dann auf,

- wenn unvorhersehbare Ereignisse eintreten oder
- Technologiesprünge zur Verhaltensänderungen führen

Solche Situationen sind eher mit qualitativen als mit quantitativen Methoden beherrschbar, aber quantitative Methoden liefern numerische Werte, die in Modellen verwendbar sind

$\Rightarrow$ Kombination beider Vorgehensweisen ist notwendig

Vorhersagestrategie



Kombination qualitativer und quantitativer Techniken

- Quantitative Techniken liefern auf wohldefinierte Weise eindeutige Werte
- Qualitative Techniken können semi-formal verwendet werden und liefern in der Regel nur Tendenzen

Schritte der Vorhersage:

- Festlegung der Vorhersageziele
  - Was soll vorhergesagt werden?
  - Für welchen Zeitraum soll die Vorhersage erfolgen?
  - Was sind die kritischen Parameter für anstehende Entscheidungen?
- Definition der qualitativen Vorhersage
  - Welche technologischen Einflüsse wirken auf das System ein?
  - Welche neuen Entwicklungen existieren und werden den Markt in Zukunft beeinflussen?
- Analyse historischer Daten
  - Welche historischen Daten verwenden?
  - Wie feinkörnig soll die Vorhersage sein?
- Auswahl der quantitativen Vorhersagetechnik
  Abhängig von verfügbaren Daten, Vorhersagezeitraum und "subjektiven" Erwartungen
- Lastvorhersage durch Kombination quantitativer-qualitativer Resultate

4.4.2 Vom Geschäftsprozess zu den Modellparametern

Vorhersage liefert in der Regel Resultate auf Ebene der Anwendung

Für die Leistungsmodelle werden benötigt:

- Lastintensitäten beeinflusst durch Benutzerverhalten
- Bedienzeiten beeinflusst durch Benutzerverhalten und Rechnertechnologie

Entwicklung der Rechnertechnologie ist mittelfristig in der Regel gut vorhersagbar

Benutzerverhalten ist entscheidender Aspekt und sehr schlecht vorhersagbar

Benutzerbefragungen nach zukünftigen Lastintensitäten i.a. nicht hilfreich, da viele Prozesse ohne Kenntnis des Benutzers ablaufen

$\Rightarrow$ Ebene einzelner Befehle und Programme ist zu fein!

Besser geeignet: Analyse von Geschäftsprozessen

- Geschäftsprozesse bilden Basis betrieblicher Abläufe
- zu jedem Geschäftsprozess gehören EDV-Transaktionen
- damit erzeugt jeder Geschäftsprozess Last
- Geschäftsprozesse bilden Basis der Unternehmensplanung

Schritte der Vorhersage:

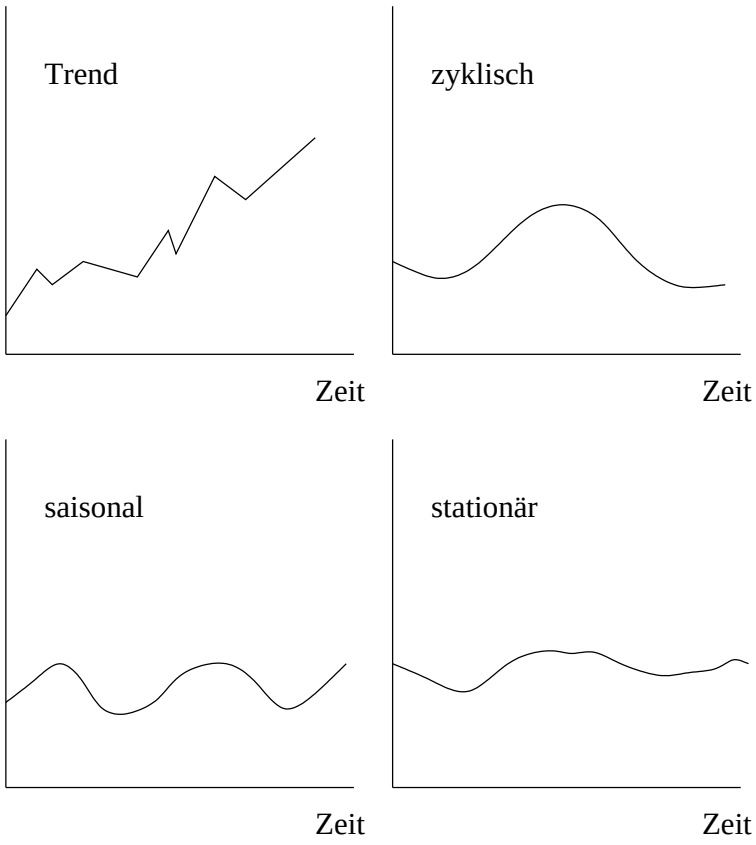
- Auswahl der Anwendungen, deren Verhalten vorhergesagt werden soll
  - Auswahl der wichtigsten Anwendungen mit hoher Systemlast
  - Zusammenfassen der restlichen Anwendungen zur Hintergrundlast
- Identifikation der Geschäftsprozesse, die diese Anwendungen benutzen
- Zusammenfassende Statistiken der ausgewählten Anwendungen
- Sammeln von Statistiken über benötigte Geschäftsprozesse
- Übersetzung von Geschäftsprozessen zu Modellparametern
  - Zwischenschritt Abbildung Geschäftsprozess auf Anwendung
- Vorhersage der Modellparameter als Funktion der Geschäftsprozesse
  - indirekt über Vorhersage der Geschäftsprozesse

4.4.3 Vorhersagetechniken

Zahlreiche Vorhersagetechniken existieren

Anwendbarkeit hängt primär von den Daten ab

Erster Schritt Visualisierung der Daten



Erste Erkenntnisse, die u.U. Techniken ausschließen

Aber Vorsicht:

Nicht alle Daten lassen sich so gut visualisieren

Skalierung und Parametrisierung kann

visuellen Eindruck stark verändern

Lineare Regression

Regressionsmodelle bestimmen den Wert einer Variablen
als Funktion anderer Variablen

- Zu bestimmende Variable: abhängige Variable
- Andere Variablen sind unabhängige Variablen

Regressionsmodelle können auf
unterschiedlichen Funktionen beruhen

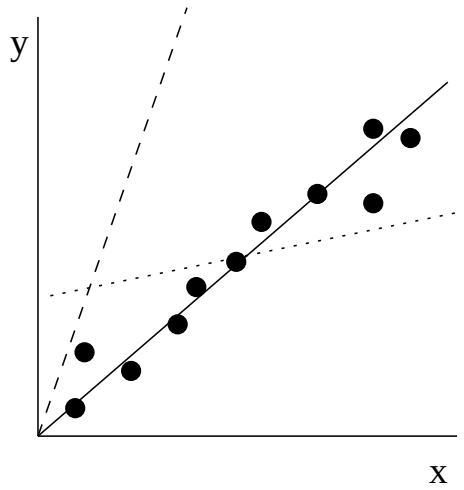
Verbreitete Form lineare Regression

$$\widehat{y} = a + b \cdot x$$

wobei $\widehat{y}$ die abhängige und x die unabhängige Variable

Gegeben Menge von n Beobachtungen x_i, y_i

Wie sind a und b zu wählen?



Wähle die Gerade, die sich gut an die Punkte anpasst
(im Beispiel die durchgezogene Linie)

Messung des Abstands der Punkte von der Geraden durch den quadrierten Abstand der y_i von den $\widehat{y}_i$ (von der Geraden)

Dies führt zu folgender Berechnung:

$$b = \frac{\sum_{i=1}^{n} x_i \cdot y_i - n \cdot \overline{x} \cdot \overline{y}}{\sum_{i=1}^{n} {x_i}^2 - n \cdot (\overline{x}^2)}$$

$$a = \overline{y} - b \cdot \overline{x}$$

mit $\overline{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$ und $\overline{y} = \frac{1}{n} \cdot \sum_{i=1}^{n} y_i$

Fehler in der Anpassung der i-ten Beobachtungen:

$$e_i = y_i - (a + b \cdot x_i)$$

Gesamtfehler der Regression werden definiert als

$$SSE = \sum_{i=1}^{n} {e_i}^2$$

a und b werden in der linearen Regression so gewählt,
dass SSE minimiert wird

Varianzbestimmung der Beobachtungen

Ohne Regression würde jedes y_i gleich $\overline{y}$ gesetzt

Fehler wäre dann

$$SST = \sum_{i=1}^{n} (y_i - \overline{y})^2$$

$SSR = SST - SSE$, der Teil der Variabilität der y_i, der nicht durch die Regression erklärt wird

Fehlervarianz $\sigma_e^2 = \frac{SSE}{n-2}$

($n-2$, da zwei Parameter bereits aus Daten geschätzt)

Falls beobachtete Daten statistischen Schwankungen unterliegen, so würde jede Beobachtung andere Werte für a und b liefern

$\Rightarrow$ Bestimmung von Konfidenzintervallen für die Regressionsparameter

Standardabweichung der Regressionsparameter

bei Unabhängigkeit der Störungen

$$\sigma_a = \sigma_e \cdot \sqrt{\left(\frac{1}{n} + \frac{\overline{x}^2}{\sum_{i=1}^{n} {x_i}^2 - n \cdot \overline{x}^2} \right)}$$

$$\sigma_b = \frac{\sigma_e}{\sqrt{\sum_{i=1}^{n} {x_i}^2 - n \cdot \overline{x}^2}}$$

Damit kann man die Konfidenzintervalle

$$a \pm t_{\alpha,n-2} \cdot \sigma_a \text{ und } b \pm t_{\alpha,n-2} \cdot \sigma_b$$

zum Niveau α bestimmen

$t_{\alpha,n}$ ist das α Quantil der t-Verteilung mit n Freiheitsgraden ist

Berechnung von Konfidenzintervallen für zukünftige Beobachtungen

Zu bestimmen

$$\widehat{y}_p = a + b \cdot x_p$$

Standardabweichung bei m Beobachtungen von y am Punkt x_p

$$\sigma_{\widehat{y}_{mp}} = \sigma_e \cdot \sqrt{\frac{1}{m} + \frac{1}{n} + \frac{(x_p - \overline{x})^2}{\sum_{i=1}^{n} {x_i}^2 - n \cdot \overline{x}}}$$

Spezialfall $m = 1$

$$\sigma_{\widehat{y}_{1p}} = \sigma_e \cdot \sqrt{1 + \frac{1}{n} + \frac{(x_p - \overline{x})^2}{\sum_{i=1}^{n} {x_i}^2 - n \cdot \overline{x}}}$$

Spezialfall $m \to \infty$ (Mittelwert der Beobachtungen)

$$\sigma_{\widehat{y}_{\infty p}} = \sigma_e \cdot \sqrt{\frac{1}{n} + \frac{(x_p - \overline{x})^2}{\sum_{i=1}^{n} {x_i}^2 - n \cdot \overline{x}}}$$

Berechnung der Konfidenzintervalle zum Niveau α

$$\widehat{y}_p \pm t_{\alpha, n-2} \cdot \sigma_{\widehat{y}}$$

Konfidenzintervall wird breiter,

wenn x_p weiter entfernt von $\overline{x}$

Annahmen der linearen Regression

1. Wahrer Zusammenhang zwischen x und y ist linear
2. Die unabhängige Variable x kann ohne Fehler beobachtet werden
3. Auftretende Fehler sind statistisch unabhängig
4. Fehler sind normalverteilt mit Mittelwert 0 und konstanter Varianz

Falls 1. verletzt ist, gibt es folgende Möglichkeiten

Multiple Regression

$\widehat{y}_i = a + b_1 \cdot x_i^1 + \ldots + b_m \cdot x_i^m$

Vorgehen ähnlich zur linearen Regression

Nichtlineare Regression

Manchmal Transformation in lineares Modell möglich

Beispiele

nichtlinear	linear
$y = a + b/x$	$y = a + b \cdot (1/x)$
$y = 1/(a + b \cdot x)$	$(1/y) = a + b \cdot x$
$y = x/(a + b \cdot x)$	$(x/y) = a + b \cdot x$
$y = a \cdot b^x$	$\ln y = \ln a + (\ln b) \cdot x$
$y = b \cdot x^a$	$\ln y = \ln b + a \cdot \ln x$

Moving Average

Sehr einfache Technik

Ein Wert in der Zukunft wird als Mittelwert
von vergangenen Werten bestimmt

Damit nur für kurzfristige Prognosen geeignet

Berechnung des Vorhersagewertes

$$\widehat{y}_{n+1} = \frac{y_n + y_{n-1} + \ldots + y_{n-t+1}}{t}$$

Wert für $\widehat{y}_{n+2}$ kann erst berechnet werden,
wenn y_{n+1} bekannt

Freier Parameter t

Wähle t $(< n)$ so das

$$\frac{\sum_{i=t+1}^{n} (y_i - \widehat{y}_i)^2}{n-t}$$

minimiert wird

Also Auswahl von t, so dass in der Vergangenheit
das beste Ergebnisse erzielt worden wäre

Exponential Smoothing

Ähnlich zu moving average nur für kurze Vorhersagen

Berechnung des Vorhersagewerts aus der letzten Beobachtung und der letzten Vorhersage

$$\widehat{y}_{n+1} = \widehat{y}_n + \alpha \cdot (y_n - \widehat{y}_n)$$

α : Anpassungsgewicht $(0 < \alpha < 1)$

Auswahl von α auf Basis bisheriger Beobachtungen

Anwendungen der Verfahren

Gewählte Vorhersagetechnik immer erst gegen die verfügbaren Daten validieren

- Parameter aus einem Teil der Daten schätzen
- Restliche Daten zur Validierung verwenden

Vorgestellte Techniken können nur für eine unabhängige Variable verwendet werden

Probleme mit mehreren unabhängigen Variablen erfordern

- Transformation des Problems oder
- komplexere Vorhersagetechniken

Eine allgemeien Aussage zum Abschluss:

"Eine richtige Prognose trifft meistens nicht ein"

(weil auf ihrer Basis Verhaltensänderungen auftreten, welche die Randbedingungen ändern)