Kapitel 2
Performance-Theorie

Wie in den späteren Kapiteln auch, werden wir die Systemseite und die Anwendungsseite getrennt behandeln. Die Praxis lehrt zwar, dass beide Aspekte eigentlich untrennbar miteinander verwoben sind und Änderungen in der Parametrisierung des einen mit an Sicherheit grenzender Wahrscheinlichkeit Effekte auf der anderen Seite nach sich ziehen werden, aus mindestens zwei Gründen jedoch eine Abschichtung des Gesamtproblems durch eine solche Separatbetrachtung erleichtert wird:

- Auftrennung von Problembereichen nach wechselseitigen Schwerpunkten
- Identifizierung spezifischer Maßnahmen (ohne mögliche Wechselwirkungen aus den Augen zu verlieren).

Eine vermengte Gesamtbetrachtung der Performance-Komplexität erschwert ungemein das Herausarbeiten von kritischen Einzelaspekten. In diesem Kapitel werden wir also die beiden Linien System und Anwendung nacheinander betrachten. Wegen ihrer unterschiedlichen Natur werden diese Abschnitte anders strukturiert sein.

2.1 System-Performance

Der Abschnitt Systemparameter teilt sich selbst noch einmal auf in

- Hardware-Parameter und
- Betriebssystemparameter.

Auch hier haben wir wiederum Verflechtungen, da beide sich bedingen. Beide Elemente machen aus, was man Systemarchitektur nennt, sollten aber, wenn möglich, getrennt betrachtet werden. Das Spektrum möglicher Betriebssystemparameter ist natürlich abhängig vom Hardwarehersteller. Auf der anderen Seite ist eine fast unendliche Kombinatorik von Betriebseinstellungen denkbar für eine gegebene Hardwarekonfiguration desselben Herstellers – nur begrenzt durch die Realitäten der laufenden Anwendungen, wobei wir wieder bei den oben genannten Randbedingungen wären. Umgekehrt kann dieselbe Version eines Betriebssystems auf unterschiedlichen Hardwareumgebungen zuhause sein. Wir werden zunächst auf die Hardware selbst eingehen.

W. W. Osterhage, *Performance-Optimierung,* X.systems.press,
DOI 10.1007/978-3-642-17190-1_2, © Springer-Verlag Berlin Heidelberg 2012

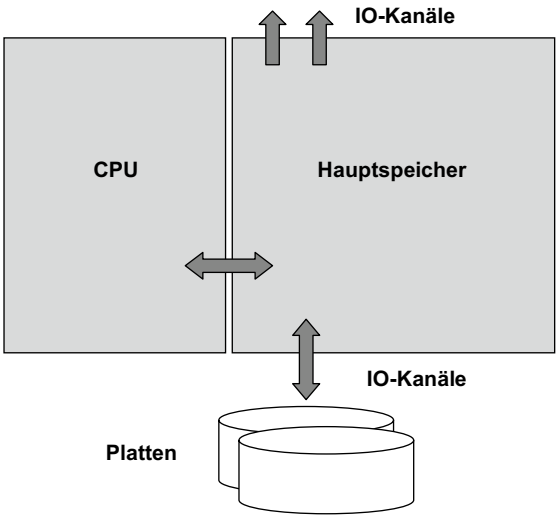

Abb. 2.1 Performance-Systemobjekte

2.1.1 Hardware-Parameter

2.1.1.1 Allgemeine Hinweise

Eigentlich müsste es besser heißen: System- bzw. Hardware-Komponenten. Zu denen, die losgelöst vom Gesamtsystem betrachtet werden können, gehören (Abb. 2.1 wie Abb. 1.2):

- CPU
- Hauptspeicher
- Plattenspeicher
- Ein-/Ausgabe-Kanäle.

Selbstverständlich gehören zur Hardware noch viele andere Elemente, wie Endgeräte, Modems und andere Kommunikationskomponenten, die aber für unsere Performance-Betrachtungen hier nicht gesondert abgehandelt werden sollen (z. B. Konfiguration der Cursor-Geschwindigkeit durch Mauseinstellungen). Die Theorie der System-Performance behandelt die oben genannten Komponenten in dieser Reihenfolge. Das Gewicht der jeweiligen Komponente bezogen auf eine bestimmte Performance-Situation hängt von der Art der Anwendung, der Anzahl User und weiteren Faktoren ab. Deshalb sollte man im Vorfeld keine Prioritätenfestlegung treffen.

Auch zwischen diesen Komponenten gibt es Beziehungen, die jeweils ebenso aufgezeigt werden. Obwohl die erwähnten Ressourcen zunächst isoliert betrachtet werden, hängt die Bedeutung für spezifische Performance-Situationen natürlich wiederum – wie bereits erwähnt – von den Anwendungen ab, die diese Ressourcen in Anspruch nehmen. Zur Einkreisung eines spezifischen Problems sind dennoch

zunächst Aussagen erforderlich, die grundsätzliche Mechanismen offenlegen, die unabhängig von den jeweiligen Anwendungsfällen Gültigkeit besitzen.

Bevor wir uns nun den Ressourcen und deren Eigenheiten im Detail zuwenden, sollen noch einige allgemeine Grundsätze angesprochen werden. Dazu gehören auch die Fragen:

- Wie kann System-Performance grundsätzlich getestet werden? und
- Wann macht Performance-Messung Sinn?

Performance-Testung ist eine Disziplin, über die man auf Menschen, Prozesse und Technologie Einfluss nehmen kann, um Risiken zu vermeiden, die z. B. bei Systemeinführung, Upgrades oder Patch-Einspielung entstehen können. Kurz gesagt: Performance-Tests bestehen dann darin, eine typische Produktionssystemlast zu erzeugen, bevor z. B. neue Anwendungen eingespielt werden, um das Leistungsverhalten zu messen, zu analysieren und End-User-Erfahrung zu sammeln. Ziel ist es also, Performance-Probleme unter Produktionsbedingungen zu identifizieren und zu beheben. Ein gut vorbereiteter Performance-Test sollte in der Lage sein, die folgenden Fragen zu beantworten:

- Ist das Antwortzeitverhalten für die End-User zufriedenstellend?
- Kann die Anwendung die voraussichtliche Systemlast bewältigen?
- Kann die Anwendung die Anzahl Transaktionen, die durch die Geschäftsvorfälle zu erwarten sind, bewältigen?
- Bleibt die Anwendung stabil unter zu erwartenden, aber auch unerwarteten Lastzuständen?
- Werden End-User beim Scharfschalten positiv überrascht sein oder eher nicht?

Indem diese Fragen beantwortet werden, hilft Performance-Testung, die Auswirkungen von Veränderungen und die Risiken von Systemeinführungen zu kontrollieren. In diesem Sinne sollten Performance-Tests folgende Aktivitäten be-inhalten:

- Emulation von Dutzenden, Hunderten oder gar Tausenden von End-Usern, die mit dem System interagieren
- konsistente Last-Wiederholungen
- Antwortzeitverhalten messen
- Systemkomponenten unter Last messen
- Analyse und Abschlussbericht.

Tests und Analyse sollten stattfinden, bevor Engpasssituationen auftreten. Ansonsten müssen später Performance-Messungen am laufenden System durchgeführt und entsprechende Maßnahmen ergriffen werden. Das bedeutet, dass man die Lastverteilung über den Arbeitstag bereits im Vorfeld abschätzen möchte, oder auch, welche Jobs wann laufen dürfen. Finden sich nach Einführung dann tatsächlich Durchsatzprobleme, bietet sich folgende allgemeine Vorgehensreihenfolge an (Abb. 2.2):

- zunächst CPU unter Last messen
- dann Speicherauslastung
- Sind CPU und Speicher problemlos, sollten die Ein-Ausgabevorgänge zu den Plattenspeichern untersucht werden.

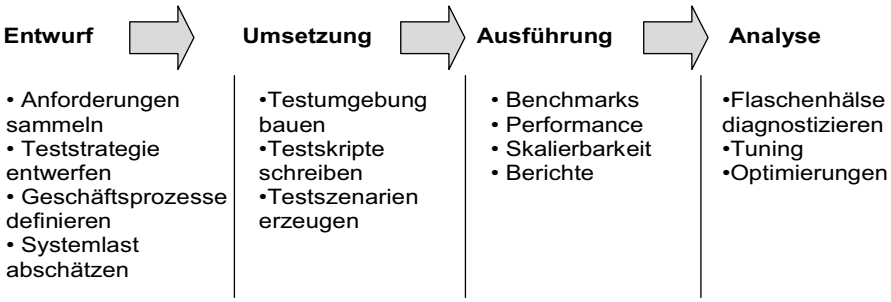

Abb. 2.2 Schritte zur Performance-Optimierung. (Nach HP Dokument 4AA1–4227ENW)

Neben der bisher diskutierten vorausschauenden Performance-Beobachtung sollten kontinuierlich oder in regelmäßigen Abständen Messungen am laufenden System erfolgen – insbesondere dann, wenn subjektiv der Verdacht schlechter Performance gemeldet wird: lange Antwortzeiten im Dialog, lang laufende Auswertungen.

Ein Problem dabei ist natürlich die anzusetzende Messlatte. Das subjektive Empfinden eines vor einem stummen Bildschirm sitzenden Nutzers, der auf die Rückkehr seines Cursors wartet – also das reine Antwortzeitverhalten – ist meistens nicht ausreichend, um als messbare Größe zu gelten. In der Regel werden deshalb neben den Online-Funktionen oder GUIs zusätzlich Batch-Läufe, deren Exekutionsdauer sich quantifizieren lässt, herangezogen, um ein Bild über den Gesamtzusammenhang zu bekommen.

Leistungstests stellen also ein Werkzeug dar, mit dem sich feststellen lässt, ob eine Systemanpassung zu einer Verbesserung oder Verschlechterung führen wird. Dabei darf die Konzeptionierung solcher Tests in der Vorbereitung nicht unterschätzt werden. Sie hängt überdies von den konkreten Verhältnissen vor Ort ab:

- Anzahl User
- erwarteter Durchsatz
- vorhandene Hardware usw.

Wichtig ist auf jeden Fall, dass nicht einzelne Anwendungen isoliert getestet werden, sondern reale Bedingungen entweder simuliert oder wie vorgefunden betrachtet werden, bei denen viele Systemprozesse parallel laufen.

Für den einzelnen Anwender reduzieren sich Leistungstests auf ganz handfeste Kennzahlen:

- Antwortzeit und
- Verweilzeit.

Das sind Größen, die direkte Auswirkungen haben auf den Arbeitsfortschritt und damit auf die persönliche Produktivität unter Zuhilfenahme der IT-Infrastruktur. Die Prozessschritte setzen sich insgesamt aus der reinen Bedienzeit und einer eventuellen Wartezeit auf Systemantworten bzw. -verfügbarkeiten zusammen. Wartezeiten sind eine direkte Konsequenz der jeweils aktuellen Systemauslastung. Genauso hat also Performance-Tuning einen direkten Effekt auf die Produktivität.

Demgegenüber stehen die Erwartungen des operativen Betriebs, die in eine andere Richtung gehen. Das Interesse hier orientiert sich an der Durchsatzmaximierung auf der Job-Ebene – also eben an der Erzeugung hoher Auslastungen. Beide Erwartungshaltungen – die des Anwenders und die des operativen Betriebs – müssen beim Tuning kompromisshaft zusammengeführt werden. Dafür stehen gewisse Richtwerte zur Verfügung, die fallweise unterstützend herangezogen werden können. Dabei handelt es sich nicht allein um quantitativ-technische Werte, sondern um Abwägungen, die insgesamt eine ineffiziente Nutzung von System-Ressourcen verhindern sollen. Vor den ersten Maßnahmen der Performance-Analyse sind folgende Klärungen sinnvoll:

- Geht es um reine Durchsatzprobleme?
- Ist die Performance des Gesamtsystems unzureichend?

Quantitativ lassen sich dazu folgende Eingrenzungen machen:

- Transaktionsraten in Abhängigkeit vom Antwortzeitverhalten
- Durchsatzraten in Abhängigkeit von Job-Verweilzeiten

Auf den Anwender bezogen gibt es weitere Kenngrößen:

- Antwortzeiten bezogen auf den Transaktionstyp
- Verweilzeiten für beauftragte Jobs

Das Leistungsverhalten des Systems lässt sich nun wiederum auf seine einzelnen Komponenten umlegen; andererseits können sich auch organisatorische Schwächen der Produktionssteuerung dahinter verbergen, wenn eine unausgewogene Job-Steuerung dahintersteckt. Beim Einsatz von Messsystemen ist zu berücksichtigen, dass solche Systeme ihrerseits auch wieder Ressourcen verbrauchen.

Ein wichtiger Richtwert bei der Messung ist die sogenannte Systemzeit. Sie setzt sich aus folgenden Anteilen zusammen: Zeiten zur

- Steuerung des Timesharing mehrerer gleichzeitiger Prozesse
- Steuerung von Ein-Ausgaben
- Steuerung des Swappings von Hauptspeicherbeladungen

Aus der Erfahrung kann man für die Systemzeit folgende Richtwerte für die CPU festlegen:

- Systemzeit insgesamt: 10–20 %
- Timesharing: 5–10 %
- Ein-Ausgaben: 2–6 %
- Swapping: 1–2 %

Abbildungen 2.3 und 2.4 zeigen grundsätzliche Vorgehensweisen im Ablauf eines Tuning-Prozesses. Zum Schluss dieser Vorrede noch zwei generelle Hinweise: Aufrüstung vorhandener Hardware ist meistens nicht das Allheilmittel bei Performance-Engpässen, wie weiter unten sichtbar werden wird. Und: Performance- Probleme wachsen exponentiell mit der zu bewältigenden Datenmenge. Nach diesen grundsätzlichen Vorbetrachtungen folgt nunmehr die Einzelbetrachtung der fraglichen Ressourcen.

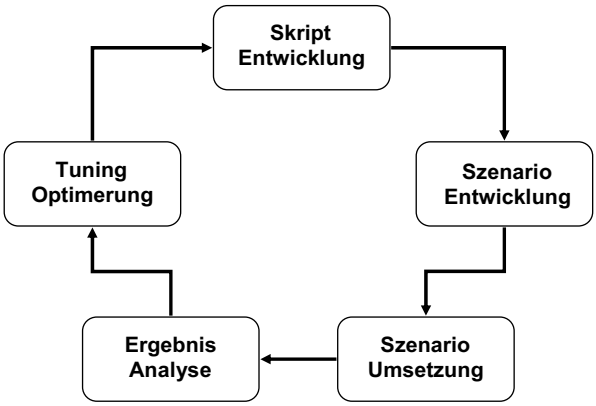

Abb. 2.3 Optimierungszyklus. (Nach HP-Dokument 4AA1–5197ENW)

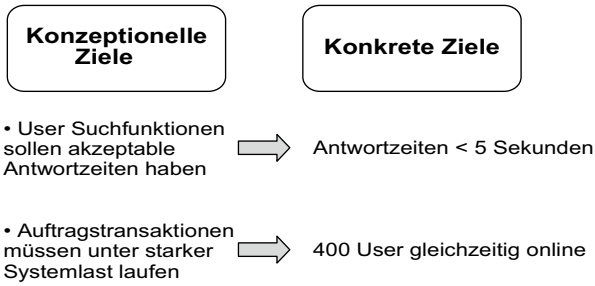

Abb. 2.4 Von der allgemeinen Konzeption zu konkreten Zielen. (Nach HP-Dokument 4AA1–5197ENW)

2.1.2 CPU

CPU steht für Central Processor Unit. Unter dem Gesichtspunkt der Performance geht es bei der CPU-Betrachtung um Leistung, d. h. Durchsatz über den Prozessor. Gemessen wird diese Leistung in mips: million instructions per second. Dabei handelt es sich um eine reine Papiergröße, die etwas über nutzbare Leistung aussagt und über Overheads und Task-Verarbeitung ohne Eingabe-/Ausgabe-Vorgänge oder Warteschlangenmanagement für nicht verfügbare andere Ressourcen, auch nicht virtual Memory Management.

CPU-Leistung wird ein kritischer Faktor für Anwendungen, die stark CPU-gebunden sind, wie wissenschaftliche und technische Anwendungen mit langen Sequenzen mathematischer Berechnungen.

Eine Messgröße kann dabei der „Relative Performance Faktor“ (RPF) (Tab. 2.1) sein. Dieser Wert macht eine Aussage über die Leistungsfähigkeit einer CPU – und zwar unter möglichst realen Produktionsbedingungen. Zu seiner Ermittlung werden sowohl Online-Anwendungen als auch Batch-Verarbeitung hinzugezogen. Bei ersteren wird die Transaktionsrate als Maß, bei letzterer die Verarbeitung von Jobs pro Zeiteinheit berücksichtigt. Je nach Austarierung der Anwendungslandschaft werden

Tab. 2.1 RPF-Ermittlung

CPU	RPF-Batch	RPF-Online	RPF-gewichtet
–	–	–	–
–	–	–	–

Tab. 2.2 Erhöhung Management-Overheads in Multipler CPU-Umgebung. (Nach Siemens: BS 2000/ OSD Performance-Handbuch Nov. 2009)

Anzahl CPUs	Erhöhung CPU-Zeit
2	5–10 %
4	10–15 %
6	15–20 %
8	20–30 %
10	25–35 %
12	30–40 %
15	35–45 %

diese Maße gewichtet, z. B. 20 % für Batch, 80 % für Online. Mit einer einzigen Zahl kann man allerdings keine schlüssigen Aussagen über das CPU-Verhalten insgesamt machen. In der Regel ist die Anwendungslandschaft dazu zu komplex. Weitere Einflussfaktoren sind durch die Serverarchitekturen gegeben. All diese Überlegungen machen natürlich nur Sinn, wenn CPU-Performance und Hauptspeichergröße sich von vornherein in einem vernünftigen Verhältnis zueinander befinden.

Eine CPU kann sich in folgenden Zuständen befinden:

- User busy
- Overhead-Verarbeitung
- im Wartezustand
- idle.

User busy bezieht sich auf das Ausführen von Anwendungstasks; Overhead-Verarbeitung deutet darauf hin, dass sich die CPU mit sich selbst beschäftigt, z. B. indem sie durch die Warteschlangen geht oder Prioritäten neu verteilt. Wartezustand weist darauf hin, dass eine benötigte Ressource nicht verfügbar ist, und „idle“ heißt, dass zurzeit keine Anforderungen an die CPU vorliegen. Die Overheads kann man noch weiter spezifizieren:

- Memory Management/Paging
- Process Interrupt Control
- Cache Management.

Ganz anders sieht das Problem in einer Multiprozessorenumgebung aus (Tab. 2.2). Diese Umgebung konstituiert sich einmal aus den Verarbeitungsprozessoren, andererseits auch aus einer dazu im Verhältnis geringeren Anzahl von I/O-Prozessoren. Um diese Konfigurationen optimal auszunutzen, müssen die Anwendungen entsprechend geschrieben sein. Dazu gehört unter anderem Parallelverarbeitung ohne gegenseitige Interferenzen.

Davon zu unterscheiden sind Konfigurationen, die Prozessoren unter dem Gesichtspunkt der Ausfallsicherheit parallel betreiben, um eine kontinuierliche Verfügbarkeit von Systemen zu gewährleisten (Abb. 2.5).

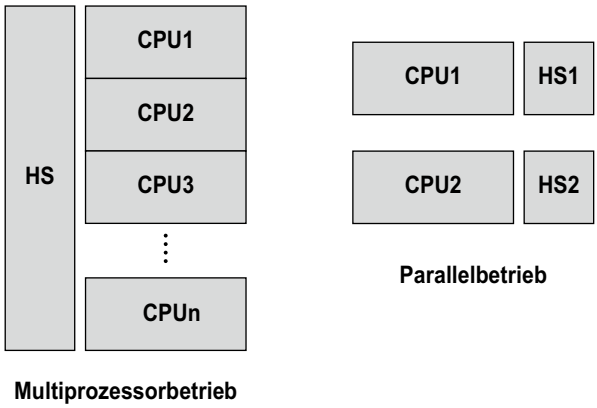

Abb. 2.5 CPU-Betriebsarten

Tab. 2.3 Verbesserungseffekt durch CPU-Zuschaltung. (Nach Siemens: BS 2000/OSD Performance-Handbuch Nov. 2009)

Anzahl CPUs	Faktor
2	1,7–1,9
4	3,3–3,6
6	4,6–5,2
8	5,8–6,6
10	6,8–7,8
12	7,8–8,8
15	8,8–9,9

Auf der Ebene der Tasks kann einer bestimmten Task immer nur ein Prozessor zugeordnet werden und umgekehrt. Daraus ergibt sich schon, dass durch reine Hardware-Aufstockung (mehr Prozessoren) Performance-Probleme nur bedingt gelöst werden können, was das Antwortzeitverhalten betrifft. Im Grenzfall findet lediglich eine schnellere CPU-Zuteilung, aber keine sonstige Durchsatzverbesserung statt. Ebenso ist eine signifikante Beschleunigung von Batch-Läufen nur durch massive Parallelität zu erreichen. Daneben ist zu beachten, dass mit steigender Prozessorzahl die Anforderungen an die Synchronisation mit dem Hauptspeicher steigen und dadurch zusätzliche Overheads erzeugt werden. Tabelle 2.3 zeigt beispielhaft für Siemens-Systeme unter BS 2000 die Verbesserungsmöglichkeiten für die Verarbeitung unabhängiger Tasks durch Zuschaltung mehrere CPUs.

Solche Upgrades haben in der Regel zur Folge, dass auch Hauptspeicher und Laufwerke angepasst werden müssen.

Viele Hersteller geben Richtwerte bezüglich der optimalen Auslastung ihrer CPUs. Betrachtet werden im Weiteren lediglich Monoprozessoren. Bei Online-orientierten Systemen sollte die CPU-Auslastung durch die Hauptanwendungen 70 % nicht übersteigen. Darüber hinaus kann es zu Problemen beim Management der Warteschlangen und den zugehörigen Wartezeiten kommen. Naturgemäß gehen die möglichen Auslastungszahlen in Multiprozessorumgebungen über diesen Richtwert hinaus (Tab. 2.4):

Tab. 2.4 Auslastung in Abhängigkeit der Anzahl CPUs. (Nach Siemens: BS 2000/OSD Performance-Handbuch Nov. 2009)

Auslastung	Anzahl CPUs
75 %	2
80 %	4
85 %	6
90 %	8

Um die Tasks in der CPU zu steuern, werden bestimmte Routinen eingesetzt, welche managen und überwachen. Folgende Parameter spielen beim Task-Management eine Rolle:

- Aktivierung
- Initiierung
- Multi-Programming Level
- Priorität
- Multiprogramming-Level (MPL) pro Kategorie
- Betriebsmittel-Auslastung (CPU, Hauptspeicher, Paging-Aktivität)
- Systemdienste (CPU-Zeit, Anzahl Ein-/Ausgaben)
- Zuteilung der Berechtigung zur Hauptspeicher-Nutzung
- Zuteilung des Betriebsmittels CPU
- Zustand „active, ready“
- Deinitiierung
- Betriebsmittelentzug bei Wartezeiten
- Zustand „active, not ready“
- Verdrängung

Da die CPU von vielen Tasks gleichzeitig genutzt wird, ist die Auslastung ein wichtiges Kriterium für Wartezeiten. Die durchschnittliche Bedienzeit durch die CPU ist allerdings klein im Verhältnis zu den Ein- und Ausgabezeiten. Das bedeutet, dass das Antwortzeitverhalten wesentlich von Ein-/Ausgabevorgängen, insbesondere durch das Lesen und Schreiben auf Speichermedien beeinflusst wird. Dadurch kommen die bereits genannten 70 % Auslastung zustande. Einige Hersteller erlauben zudem eine manuelle Prioritätenvergabe, sodass auch ursprünglich niedrig priorisierte Tasks aus den Warteschlangen heraus zum Zuge kommen können. Eine 100 %ige Auslastung ist ein idealer Wert, der in der Praxis nicht erreicht wird.

Prioritäten beziehen sich auf den Zugriff auf CPU-Ressourcen, I/Os sind dabei nicht berücksichtigt. Im Normalfall sind Prioritäten so vergeben, dass Online-Tasks grundsätzlich höher priorisiert sind als Batches. Systemprogramme wiederum haben meistens eine höhere Priorität als die übrigen Online-Funktionen. Innerhalb der Programmverarbeitung berechnet das System selbst dann die Prioritäten nach einem Algorithmus, der Warteschlangen-Position, Swap-Rate und andere Parameter einbezieht. Die Vergabe einer externen fixen Priorität ist möglich, sollte jedoch mit Vorsicht gehandhabt werden, da die Priorität sich danach nicht mehr dynamisch anpasst und je nach Einstellung zu Verdrängungen anderer Prozesse führen kann. So kann man grundsätzlich wenigen Nutzern, die große Programme fahren, niedrigere Prioritäten geben als den vielen anderen Usern, die nur kleine Tasks erledigen

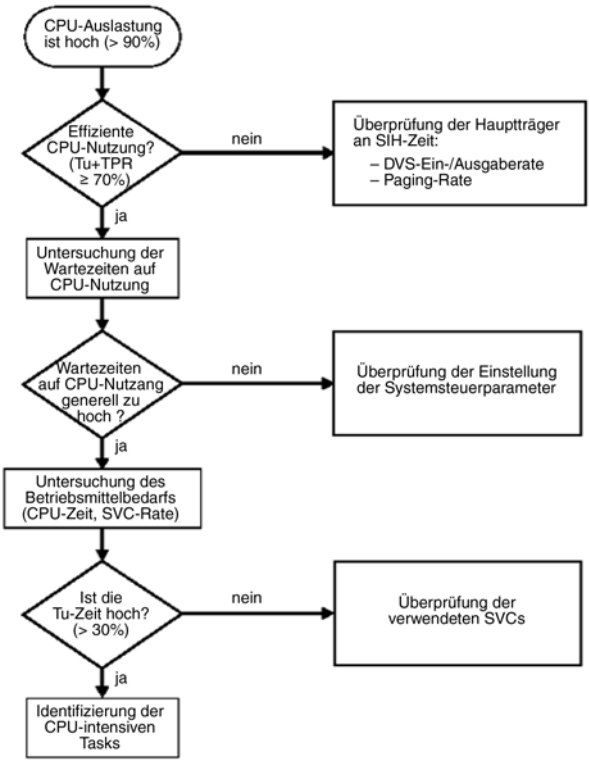

Abb. 2.6 Tuning-Ansätze bei hoher CPU-Auslastung. (Nach Siemens: BS 2000/ OSD Performance-Handbuch Nov. 2009)

wollen. Kurzfristig kann man bei eiligen Geschäftsvorfällen die Priorität erhöhen, ebenso für speicherintensive Programme, die alles blockieren, um das System wieder freizuschaufeln (Abb. 2.6).

Spricht man von einer hohen CPU-Auslastung, dann liegt diese bei über 90 %. Das braucht noch keine Engpasssituation zu sein. Bei hoher Auslastung ist es allerdings besonders wichtig, die Effizienz des CPU-Betriebes zu beobachten. Dazu sind folgende Maßnahmen von Bedeutung:

- Überprüfen der Höhe der Ein-/Ausgaberate
- Überprüfen der Höhe der Paging-Rate

Ein Maß für die Effizienz einer CPU bei hoher Auslastung ist das Verhältnis Wartezeit zu Benutzung der CPU/Hardware-Bedienzeit der CPU.

Beim Auftreten längerer Wartezeiten sind folgende Ursachen möglich:

- suboptimale Einstellung von Systemparametern
- Gesamter CPU-Zeitbedarf für alle Tasks ist zu hoch. Eine Analyse der Warteschlangenzeiten kann darüber Aufschluss geben.

Eine wichtige Rolle beim Mechanismus der CPU-Verwaltung ist der Umgang mit Unterbrechungen. Offensichtlich finden beim Swapping Unterbrechungen von exe-

kutierendem Code statt. Dadurch wird ein Programm oder ein Programmteil angehalten, das Code-Segment aus dem Hauptspeicher ausgeladen und die Task in eine Warteschlange verwiesen, aus der sie entsprechend ihrer Priorität (im Normalfall: Position in der Schlange) später wieder Ressourcen bekommt. Unterbrechungen können folgende Ursachen haben:

- prioritärer Ressourcen-Zugriff durch das Betriebssystem
- Abschluss eines geschlossenen Programmteils
- auftretende technische Fehler („exception“)
- längere Pause bei der Adressierung der Programmseite
- I/Os
- fehlende Daten
- Status „idle“ mit timeout

Es können aber auch bei niedriger CPU-Auslastung Performance-Probleme in Erscheinung treten. Dafür können folgende Gründe verantwortlich sein:

- Ein-/Ausgabekonflikte
- hoher Zugriff auf Speichermedien durch gleichzeitig viele User (zentrale Dateien)
- zu hohe Paging-Rate
- Engpässe durch Server-Tasks
- ineffiziente Parallelverarbeitung durch zu wenig Tasks
- CPU-Zeit für das Ausführen bestimmter Programme
- zu viele Systemaufrufe
- Systemselbstverwaltung
- zu viele Bibliotheksroutinen
- zu viele I/O-Routinen
- zu viele gleichzeitige User
- Multiprozesser-Management
- ungünstige CPU-Taktzahl pro Prozess
- Prozessor-Stillstand
- Abfolge von Maschinenbefehlen.

Falls Performance-Probleme weiter bestehen, selbst wenn die CPU „idle“ ist, so sind diese Probleme woanders zu suchen. Durch eine Aufrüstung der CPU ist dann nichts gewonnen.

2.1.3 Hauptspeicher

Ein wesentliches Merkmal für den Hauptspeicher ist seine nominelle Größe, nämlich die Anzahl von Zeichen, Bytes, die für eine zentrale Verarbeitung und die dazu erforderliche temporäre Speicherung zur Verfügung stehen. Die absolute Größe des Hauptspeichers kann für eine Performance-Situation

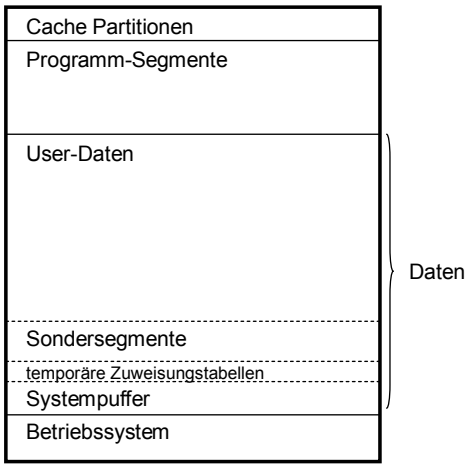

Abb. 2.7 Hauptspeicherbelegung

- passend
- zu groß oder
- zu klein

sein. Das liest sich banal. Dabei gibt es Folgendes zu bedenken: Ein überdimensionierter Hauptspeicher, häufig als Empfehlung bei Engpässen ausgesprochen, kann zu sekundären CPU-Problemen führen, wenn die Leistung nicht Schritt hält, oder eine Unterdimensionierung von I/O-Kanälen aufweisen. Der Grund liegt darin, dass die Anzahl möglicher Tasks, die geladen werden können, die Möglichkeiten dieser beiden anderen Ressourcen übersteigt, sodass sich Warteschlangen herausbilden.

Natürlich ist ein zu klein dimensionierter Hauptspeicher eher das Problem. Der Hauptspeicher hat umzugehen mit (Abb. 2.7):

- Programm-Segmenten
- Daten
- Cache-Partitionen und
- Teilen vom Betriebssystem

Normalerweise ist ein Teil des Hauptspeichers ständig durch Grundfunktionen des Betriebssystems belegt: base memory mit einer fixen Prozentzahl des gesamten Speichers, z. B. 10 %. Weitere Anteile des Betriebssystems werden später je nach Bedarf nachgeladen. Ein gewichtiger Teil wird für die User-Prozesse benötigt. Deshalb ist z. B. bei Betriebssystem-Updates, die zusätzliche Funktionalitäten beinhalten können, darauf zu achten, dass entsprechend neu und sparsam konfiguriert wird.

Cache-Partitionen werden benötigt für Speichersegmente, die Daten enthalten, auf denen wiederholt zugegriffen wird. Diese Partitionen sind konfigurierbar, können aber auch dynamisch zugewiesen werden. Im letzteren Fall stellt die Cache kein besonderes Problem dar, obwohl bei Engpässen Einschränkungen bezüglich des Cache-Anteils entstehen können. Die Performance verschlechtert sich wiederum dadurch, dass die Daten jetzt wieder direkt von den Platten gelesen werden müssen. Auf der anderen Seite führt eine intensive Cache-Nutzung zu Interrupt und Adressen-Mapping. Dadurch entstehen zusätzliche CPU-Overheads.

Die Datenbelegung des Hauptspeichers speist sich aus folgenden Quellen:

- Userdaten
- temporäre Zuweisungstabellen
- Sondersegmente
- Systempuffer.

Die User-Daten beaufschlagen den einem spezifischen User zugewiesenen Adressraum, z. B. für Sortiervorgänge. Sondersegmente werden beispielsweise für Nachrichtendateien oder Kommunikationspuffer benötigt und Systempuffer für bestimmte Systemtasks. Der Rest steht dann Programmsegmenten oder für das Paging zur Verfügung.

Ein wichtiger Faktor bei der Betrachtung von Hauptspeicher-Performance ist die Paging-Rate. Mit Paging-Rate ist die Anzahl von Vorgängen pro Zeiteinheit gemeint, die aktuell Speicher-residente Code-Seiten (Pages), die momentan für die Ausführung eines Programms durch die CPU benötigt werden, temporär aus dem Speicher entfernen, durch andere ersetzen und die ursprünglichen Seiten später wieder hineinladen. Dieser Vorgang wird auch als Swapping bezeichnet.

Es gibt einige Algorithmen, die diejenigen Hauptspeichereinträge identifizieren, die dem Swapping unterliegen sollen. Normalerweise werden die Ressource-Warteschlangen bei jedem Systemzyklus innerhalb der Frequenz der Systemuhr abgefragt, um festzustellen, auf welche Segmente während des vorhergehenden Zyklus nicht zugegriffen wurde. Diese werden dann mit einem Flag als Overlay- Kandidaten versehen. Beim nächsten Zyklus werden die Overlay-Kandidaten nochmals geprüft und dann unter Umständen geswappt, falls Speicherplatz benötigt wird. Gibt es viele Overlay-Kandidaten und lange Verweilzeiten für bestimmte Segmente, die ohne Prozessunterbrechung geswappt werden können, entstehen nur minimale Memory Management Overheads.

Hohe Paging-Raten können kritisch für eine effiziente Nutzung der Ressource Hauptspeicher werden (Abb. 2.8). Gründe dafür können sein:

- zu viele Prozesse
- zu große Programme.

Deshalb geben Hersteller auch maximal empfohlene Werte an. Memory Management für eine sauber laufende CPU sollte nicht mehr als wenige Prozent betragen. Ansonsten führen zu hohe Paging-Raten grundsätzlich zu schlechtem Antwortzeitverhalten, weil CPU-Ressourcen dafür benötigt werden. Hintergrund: In den beiden genannten Fällen versucht die CPU zunächst, die Hauptspeicherbelegung neu zu organisieren, indem ein zusammenhängender Adressraum gebildet wird (das führt zu erheblichen Overheads). Falls das nicht gelingt, gibt die CPU auf, d. h., es kommt zu Unterbrechungen von Prozessen mit niedriger Priorität, Geswappt wird in den virtuellen Speicher oder auf Paging-Bereiche der Festplatten. Konsequenzen:

- entgangene CPU-Ressourcen für Memory Management (Erhöhung der subjektiven Antwortzeiten)
- unterbrochene User-Prozesse (Erhöhung der subjektiven Antwortzeiten)

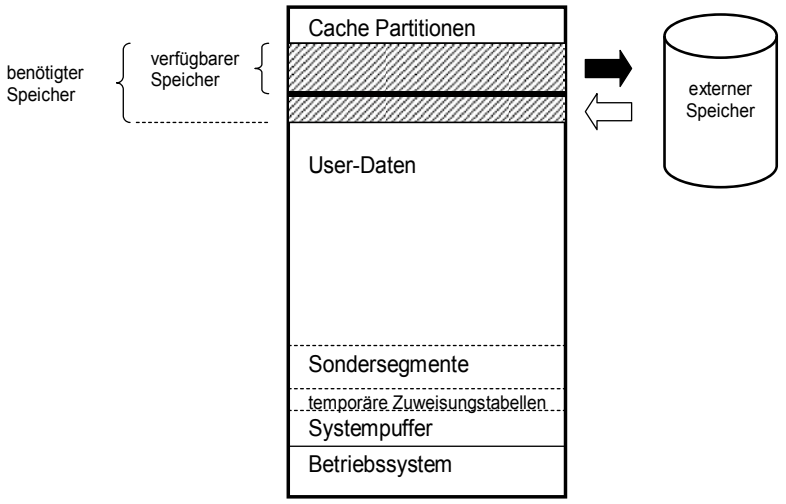

Abb. 2.8 Paging

Wird dieses Problem chronisch, geht nichts um einen Upgrade des Hauptspeichers herum. Ein Grund dafür ist auch, dass externe Swapping-Bereiche gesondert konfiguriert werden müssen. Sind diese zu klein, wird Paging zu einem Sekundärengpass mit zusätzlichen I/O-Wartezeiten und entsprechendem Memory Management.

Um Engpässe beim Hauptspeicher zu vermeiden, sind prophylaktisch regelmäßige Messungen mit einem geeigneten Monitor sinnvoll. Dabei sollten folgende Werte ermittelt werden:

- Anzahl der seitenwechselbaren Seiten (Größe des Hauptspeichers minus der Anzahl ständig für Systemzwecke residenter Seiten)
- Anzahl der global genutzten Seiten (Anzahl der seitenwechselbaren Seiten minus der Anzahl der frei nutzbaren Seiten)

Ähnlich wie bei der CPU gibt es auch für den Hauptspeicher Richtwerte. Auslastung bis 75 % gilt als unkritisch. Werden ständig über 90 % erreicht, ist ein Upgrade unerlässlich. Analog zur CPU gibt es ein Task-Management mit ganz ähnlichen Triggern über die Kontrollfunktionen der Hauptspeicherverwaltung mit Seitenverwaltungs-Algorithmus:

- Aktivierung
- Deaktivierung
- Zwangsdeaktivierung
- Verdrängung

Zusammenfassend können folgende Situationen entstehen:

Der Speicherbedarf ist höher als der vorhandene Speicher. Dann entsteht ein Paging-Problem. Das hat zur Folge, dass bestimmte Prozessschritte temporär auf dezentrale Speichermedien verlagert werden, was zu einem spürbaren Leistungsabfall führt. Lösungen können entweder durch Hardware-Erweiterung oder Anwendungs-

optimierung herbeigeführt werden. Organisatorisch könnte man auch die Anzahl User reduzieren, was in der Regel nicht akzeptiert wird.

Das gegensätzliche Szenario ist gegeben, wenn ausreichend Speicherplatz vorhanden ist. Zwischen diesen beiden Zuständen gibt es keine Alternativen. Dabei muss ein Programm nicht immer vollständig geladen sein, damit bestimmte Funktionen ausgeführt werden können. Programmseiten werden über deren Adressenfolgen stückweise geladen, was wiederum zu Swap-ins und Swap-outs, also Paging-Vorgängen führt. Ohnehin zeigt die Statistik, dass die meisten Programme 80 % ihrer Exekutionszeit in nur 20 % ihres eigenen Codes verbringen. Inaktive Prozesse werden hauptsächlich durch Online-Anwendungen hervorgerufen – wenn User vor dem Bildschirm sitzen und keine Eingaben tätigen, das Programm aber über die GUI dennoch aufgerufen bleibt. Zur Optimierung bezogen auf das Paging können folgende Maßnahmen herangezogen werden:

- Nutzung von gemeinsamen Subroutinen durch mehrere Programme
- Größenoptimierung von Codes
- Nutzung von gemeinsamen Bibliotheken
- Speicherkonfigurierung anhand geschätzter User-Zahl.

2.1.4 Platten

Es gibt einige einfache Fakten, die erklären, inwiefern der absolute Plattenplatz System-Performance beeinflussen kann: Sollte der gesamte Plattenspeicherplatz zu theoretisch möglichen 100 % belegt sein, wird das System zum Stillstand kommen. Es lassen sich Dateien (auch keine temporären oder Spools) weder erzeugen noch pflegen. Aber es gibt natürlich auch Probleme, die sich schon früher bemerkbar machen.

Jedes System, welches in die Nähe einer Plattenbelegung von 90 % kommt, wird Performance-Probleme zeitigen, sowohl was Antwortzeiten betrifft als auch eine sauber laufende Produktionsumgebung. Für einen problemlosen Betrieb benötigt man freien Speicherplatz für

- Sicherheitskopien
- Monatsabschlüsse oder
- Datenbankreorganisation

Deshalb sollte die Plattenbelegung nie mehr als 75 % betragen.

Normalerweise haben Benutzer und Administratoren nur begrenzten Einfluss auf die physische Lokalisierung von Dateien auf externen Speichermedien (Volumes). Dabei muss bedacht werden, dass sich auf physischen Volumes mehrere logische Volumes befinden können. Volumes werden über eine Identifikatiosnummer adressiert. Die Zugriffszeiten auf Platten sind abhängig von der Zugriffstechnologie, der Verwaltung des Speicherplatzes, aber auch davon, ob es sich um Umgebungen mit mehreren Rechnern gleichzeitig handelt. Weitere Beeinträchtigungen entstehen eventuell durch parallele Plattenspiegelung, insbesondere im Zusammenhang mit dem RAID-Konzept. RAID (Redundant Arrays of Independent Disks) geht den um-

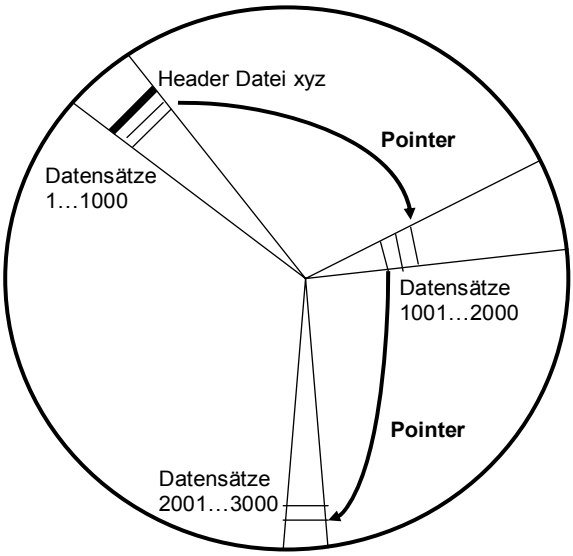

Abb. 2.9 Fragmentierung auf der Festplatte

gekehrten Weg, indem es mehrere physische Volumes unter ein logisches gruppiert. Dadurch wird die erforderliche Datenredundanz erzeugt.

Hersteller liefern Module zur Geräteverwaltung aus, die u. a. folgende Funktionalitäten beinhalten können:

- Monitoring der Plattenbelegung
- Reservierung von Plattenspeichern
- Konfiguration
- Überwachung der Zugriffsbereitschaft
- Partitionierung, um den Zugriff auf logisch zugehörige Tabellen zu optimieren

Aus all diesen Gründen ist es wichtig, im Vorfeld zu einer Neu-Installation planmäßig Speicherorte zu identifizieren, festzulegen, wo welche Daten liegen sollen, und ein entsprechendes Schema anzulegen, bevor diese Daten tatsächlich urgeladen werden.

Kompromisse müssen gefunden werden zwischen dem Ziel der Durchsatzoptimierung eines Einzelprozesses gegenüber dem Gesamtdurchsatz einer Festplatte. Will man den Einzelprozess messen, benötigt man dafür eine große Testdatei. Schwieriger ist es beim Gesamtdurchsatz einer Platte. Eigentlich geht das nur über eine Simulation, die alle Prozesse einschließt. Danach lassen sich dann Maßnahmen ergreifen, die die unterschiedlichen Interessen von Anwendungen einbeziehen. Ein weiterer Kompromiss betrifft die optimale Nutzung des Speichers gegenüber dem Durchsatz. Hier sollte man trennen: große Dateien mit wenig Speichereffizienz auf definierte Platten, viele kleinere Dateien auf andere.

Ein weiterer Aspekt betrifft die Fragmentierung von Informationen (Abb. 2.9). Es ist beileibe nicht so, dass alle Datensätze, die zu einer bestimmten Tabelle gehören, zusammenhängend auf einem Speichermedium angelegt sind – nicht einmal

zu Anfang. Dateien und Datensätze werden fraktioniert, d. h. aufgespalten und nach internen Algorithmen auf einer Platte verteilt – mit den zugehörigen Pointern, die auf die jeweiligen Folgeadressen verweisen. Im Laufe des Lebens einer Anwendung verschlimmert sich dieser Zustand dadurch, dass Tabellen durch Löschungen oder Hinzufügen von Datensätzen gepflegt werden. Dadurch entstehen einerseits Lücken auf dem Speichermedium, andererseits werden solche Lücken durch Neuanlagen wieder gefüllt. Diese Neuanlagen brauchen aber weder zu derselben Datei zu gehören, noch in irgendeiner logischen Reihenfolge zu stehen. Letztendlich verlängern sich die Suchprozesse bei Abfragen und damit die Antwortzeiten. Von Zeit zu Zeit sollte eine Defragmentierung vorgenommen werden. Sie führt dazu, dass Lücken aufgefüllt und zugehörige Daten zusammengeführt werden.

2.1.5 I/O

Wie bereits erwähnt, besteht die Gefahr, dass ein signifikanter Anteil von CPU-Ressourcen für die Verwaltung von I/O-Waits benötigt wird. Das ist insbesondere dann der Fall, wenn Prozessunterbrechungen der Grund sind. Neben den Plattenzugriffen selbst spielen dabei die Anzahl und die Geschwindigkeit von Kommunikationskanälen und deren Controller eine wichtige Rolle. Diese werden ja auch für Endgeräte, Modems und andere Kommunikationshardware benötigt. Bei einer zu geringen Anzahl von I/O-Controllern entstehen Kommunikationsengpässe. Auf der anderen Seite aber kann eine vorgegebene CPU nur eine endliche Anzahl von Controllern verwalten.

Eng zusammen mit der Performance externer Laufwerke hängt somit die Leistung der Kommunikations-Kanäle. Ein-/Ausgabevorgänge werden bei einigen Herstellern über deren interne Datenmanagement-Systeme gesteuert. Die Umsetzungszeit für Ein-/Ausgabe-Befehle dauert in der Regel eine Größenordnung länger als bei CPU-Befehlen.

Uns soll zunächst nur die Ein-/Ausgabe auf externe Speichermedien interessieren (Abb. 2.10). In den Programmen werden diese Aufrufe über Intrinsics bzw. I/O-Programmaufrufe realisiert (get, put etc.). SQL-Abfragen (read, write) werden bei ihrer Ausführung auf dazu äquivalente Ausführungsinstruktionen umgesetzt. Wesentlich bei der Ausführung solcher oder ähnlicher Abfragen ist die Struktur der dahinter liegenden Datenbanken. Suchoperationen dauern unterschiedlich lange, je nachdem, ob es sich um sequentielle Dateien, Index-sequentielle Dateien oder relationale Datenbank-Managementsysteme handelt. Exekutionszeiten hängen außerdem von der Art des Zugriffs ab. Ein rein lesender Zugriff geht naturgemäß schneller als ein Update, bei dem zuerst gefunden, dann gelesen und dann geschrieben werden muss.

Wird eine Ein-/Ausgabe für die weitere Exekution eines Programms benötigt, so wird dieser Teil des Codes zunächst in einen Wartezustand versetzt, bis der Ein-/Ausgabevorgang durchgeführt worden ist. Danach geht der Programmablauf weiter. Während des Wartezustands sitzt das Programm aus Sicht der CPU in einer

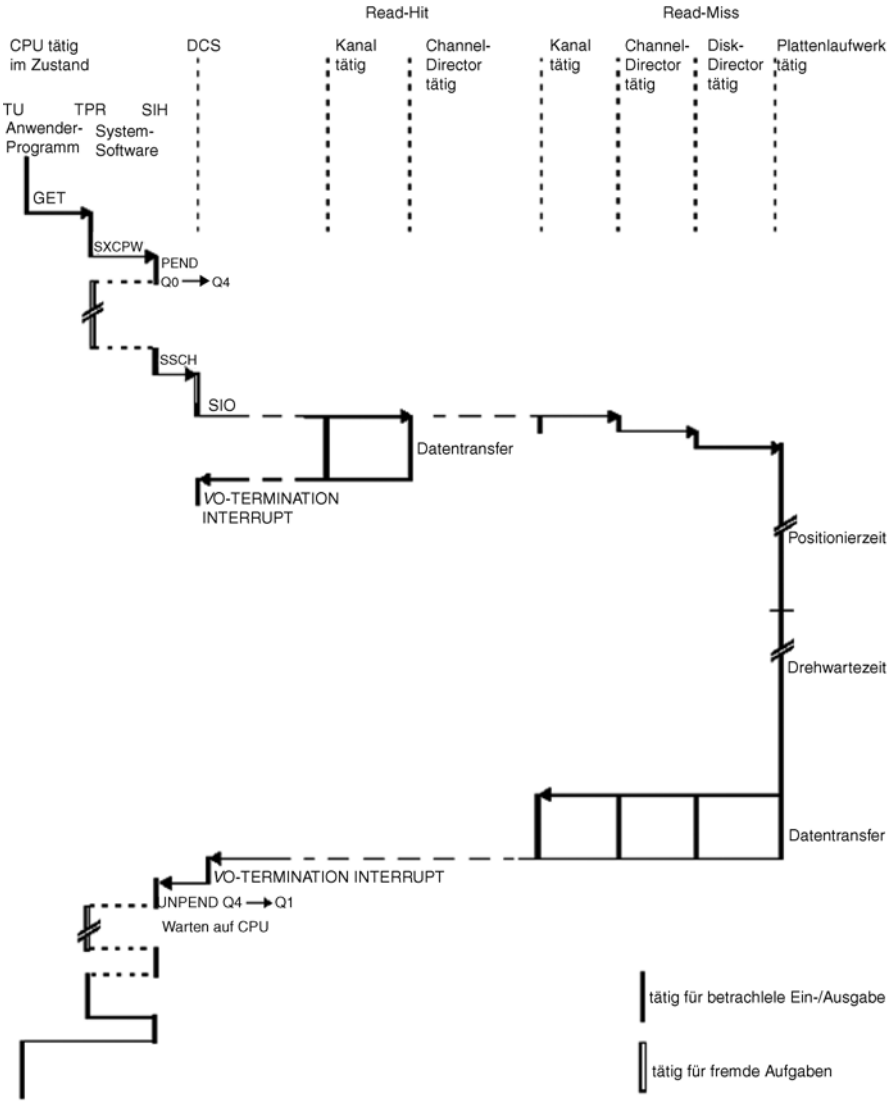

Abb. 2.10 Zeitlicher Ablauf einer Platten-Ein-/Ausgabe. (Nach Siemens: BS 2000/OSD Performance-Handbuch Nov. 2009)

entsprechenden Warteschlange, die über einen Prioritätenalgorithmus gesteuert wird. In dieser Schlange befinden sich auch Teile von anderen, konkurrierenden Programmen, sodass nach Durchführung der I/O nicht sofort sicher gestellt ist, dass der betroffene Programmteil unmittelbar weiter ausgeführt wird, sondern lediglich entsprechend der aktuellen internen Prioritäten. Erfolgt die Ein-/Ausgabe über bestimmte Kanäle bzw. Geräte, so finden sich dafür ebenfalls Warteschleifen, da auch andere User auf diese Ressourcen Zugriff beanspruchen.

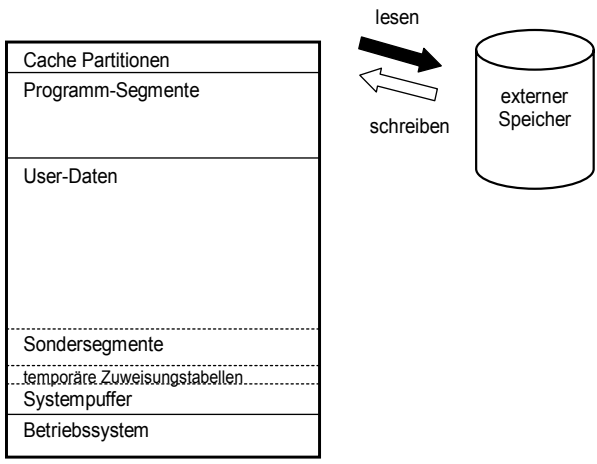

Abb. 2.11 Cache Management

Bei Ein-/Ausgaben handelt es sich also um ein komplexes Geschehen, das über diverse Warteschlangen und Datenpuffer, in die die angeforderten Datensätze eingelesen werden, zusammenspielt.

Um Zugriffe schneller zu machen, gibt es den sogenannten Daten-Cache (Abb. 2.11). Dabei handelt es sich um einen Puffer, der die zuletzt abgefragten Daten gespeichert behält, während der User oder das Programm sich mit anderen Dingen beschäftigen. Wird jetzt z. B. die letzte Abfrage aus irgendeinem Grunde wiederholt, so liegt diese Information unmittelbar im Cache vor und wird sofort zur Verfügung gestellt. Außerdem erlaubt ein Daten-Cache die Minimierung von Plattenzugriffen sowie das Bündeln von Schreiboperationen (Write-behind-Methode). Daten-Caches sind konfigurierbar. Deren Größe beeinflusst den Durchsatz. Ein Kompromiss muss gefunden werden zwischen Cache-Reservierung im Hauptspeicher, I/Os und CPU-Belastung wegen des dadurch beeinflussten Swappings.

I/O-Zeit ist außerdem abhängig von der Technologie des Systembus und der Kanalstruktur für I/Os. Bei mehreren Festplatten gibt es Optimierungskompromisse zwischen einer Gesamttransferrate über alle Platten und individuellen Transferraten einzelner Platten. Die Maximierung der Gesamttransferrate geht zu Lasten anderer, individueller Transferraten. Man kann jetzt z. B. die Anwendungen so aufteilen, dass bestimmte Platten große Dateien enthalten, andere wiederum viele kleine Dateien mit kleinen I/O-Aufträgen.

Die optimale Lösung besteht darin, dass man die Platten-I/O über alle Platten gleichmäßig verteilt. Jede Platte sollte ihren eigenen Controller besitzen. Shared Controller mit mehreren Kanäle über denselben Bus sollten vermieden werden. Insgesamt sind die Transferraten also abhängig von einer Kombination aus Platten, Bus-Performance und Controller. Eine Optimierung gelingt nicht unbedingt dadurch, dass alle Komponenten schneller gemacht werden, da neue Engpässe durch gute Performance einer Komponenten auf Kosten einer anderen entstehen können.

Die Leistungsdaten von Platten selbst sind wiederum abhängig von der Positionierzeit (Zeit der Kopfbewegung von einer Datenspur auf die nächste) in Abhängig-

keit von der Datenkontiguität. Liegt eine hohe Fragmentierung und eine Verteilung der Daten über viele Anwendungen vor, ist ein häufiges Springen des Kopfes erforderlich. Das wiederum führt zu Beschleunigungen und Abbremsungen. In einem solchen Szenario dauert der Suchvorgang selbst viel länger als das eigentliche Lesen oder Schreiben. Zur Verbesserung der Situation sollten die unterschiedlichen Partitionsmöglichkeiten, die ein Hersteller anbietet, genutzt werden.

2.1.6 Betriebssystemparameter

Jedes Betriebssystem bringt neben seinen Fähigkeiten, Anwendungen zu unterstützen, auch eigene Funktionalitäten mit, die System-Ressourcen beeinflussen und erfordern. Dazu gehören:

- Programmaufrufe
- bestimmte Befehle
- Systemsteuerungsfunktionen
- Konfigurationsparameter
- Systemtabellen
- funktionale Subsysteme: Utilities.

Bestimmte Betriebssystembefehle können Einfluss auf die Performance haben. Dazu gehören solche, die Folgendes bewirken:

- Kommunikationskontrolle
- Job limits
- Logging
- Memory-Zuweisung
- Prioritätenvergabe.

Direkten Einfluss auf die Performance haben folgende Systemsteuerungsfunktionen und Konfigurationsparameter:

- Kommunikationseinstellungen
- Job limits
- logging
- memory allocation
- priority scheduling
- Dateisystem-relevante Einstellungen
- Spool-Parameter
- time outs
- cache sizes
- Systemtabellen.

Systemtabellen bedürfen besonderer Aufmerksamkeit. Allgemein lässt sich festhalten, dass ein Tabellenüberlauf das System zum Absturz bringen wird. Wird die Tabellengröße überkonfiguriert, wird zu viel Hauptspeicher belegt.

Was die Utilities angeht, so ist auf organisatorischem Wege zu entscheiden, unter welchen Gegebenheiten bestimmten Usern oder Anwendungen welche Utilities zur Verfügung gestellt werden dürfen. Dazu gehören:

- Editoren
- Systemabfrageroutinen
- Debugging-Software
- Formatierungen
- Compiler
- Database Handler.

Daneben ist es wichtig zu prüfen, ob neue Versionen des Betriebssystems Features enthalten, die positiv zur Verbesserung der Performance eingesetzt werden können. Das sollte ein ständiger Begleitprozess sein.

2.2 Anwendungsperformance

2.2.1 Anwendungsparameter

Obwohl ein Performance-Tuning auf der Basis von Anwendungsoptimierungen meistens zeitaufwendig ist und deshalb in der Regel kurzfristig keine sichtbaren Ergebnisse zeitigt, ist es dennoch häufig unerlässlich, um mittel- und langfristige Erfolge zu erzielen. Hier sind die Einflussfaktoren:

- Programmiersprache
- modularer Programmaufbau
- Anzahl Subroutinen und externer Aufrufe
- Dateimanagement
- Ein-/Ausgabeprozeduren
- GUI oder Batch-Verarbeitung
- Größe von Codesegmenten
- Kommunikationsprozesse mit anderen Anwendungen

Falls beeinflussbar, spielt die Programmiersprache eine wichtige Rolle für die zukünftige Performance einer Anwendung. Demgegenüber sind z. B. die Realisierungszeiten abzuwägen – modularer Programmaufbau – im Gegensatz zu struktureller Programmierung, die im Zuge moderner Sprachen an Bedeutung verloren hat. Gemeint ist in diesem Zusammenhang die Entkopplung von Anwendungsmodulen untereinander unter einem gemeinsamen Frontend. Dadurch wird der Hauptspeicher entlastet. Außerdem können dabei bestimmte Module aus unterschiedlichen Anwendungsanteilen angesprochen und so gemeinsam genutzt werden – z. B. Fehlerroutinen.

Während man erwarten kann, dass jedes neue Release akzeptable Anwendungs-Performance liefert, hängt das Ergebnis doch sehr stark ab von den vorbereitenden Maßnahmen. Dabei sollten die Wünsche der User berücksichtigt werden und deren

Vorstellungen über gute Performance. Auch der gesamte Prozessablauf, in dem die IT-Unterstützung eingebunden ist, sollte bedacht werden. Am Ende sollten folgende Fragen beantwortet werden:

- Muss die Anwendungsarchitektur angepasst werden, um bestimmten Performance-Kriterien zu genügen?
- Ist ein Upgrade der IT-Infrastruktur erforderlich?
- Wird Performance negativ beeinflusst durch GUI-Gestaltung?
- Wie wird sich eine zu erwartende Durchsatzsteigerung auf die Systemlast auswirken?

Bestimmte Faktoren lassen sich manchmal mit geringem Aufwand tunen, wie z. B. Ein-/Ausgabeprozeduren oder der Wechsel von GUI auf Batch. Andere erfordern größere Investitionen. Dazu gehören sicherlich die Transposition in andere Programmiersprachen oder Änderungen am Datenmodell.

Zusätzlich zu den Messwerten, die für die reine System-Performance von Bedeutung sind, gibt es eine Reihe von Kenngrößen, die im direkten Bezug zu den Anwendungen stehen. Dazu gehören insbesondere die Task-spezifischen Betriebsmittelanforderungen im zeitlichen Durchschnitt und auf Spitzen, bezogen auf

- CPU-Zeit
- Ein-/Ausgaben inklusive Zugriffe auf bestimmte Datentabellen und Plattenzugriffsstatistiken
- Wartezeiten
- Paging-Raten
- Kanal-Transporte inklusive TCP/IP-Verbindungen
- eventuelle Kommunikationsereignisse im Rechnerverbund
- Zugriffe auf Hauptspeicher
- Antwortzeitenstatistik
- Cache-Trefferquoten
- Anzahl Job-Aufrufe

Diese Informationen werden entweder durch Messmonitore geliefert oder sind teilweise aus Logfiles zu ermitteln, wenn die entsprechenden Transaktionstypen dafür eingeschaltet sind. Logdateien für die Aufzeichnung von Transaktionen bedürfen in zweierlei Hinsicht spezifischer Ausrichtung:

- Formatierung
- Selektion.

Bei der Formatierung ist darauf zu achten, dass die Einträge möglichst von technischen Informationen befreit werden. Das heißt, alles, was Bezug zum Transaktionscode und zu Datenverschlüsselungen, Formatangaben usw. hat, sollte ausgespart werden. Lediglich aussagekräftige Textinformationen sollten hinterlegt werden, die dem Auswerter sofort einen Hinweis auf das tatsächliche Transaktionsgeschehen geben.

Die Selektion zur Aufzeichnung von Transaktionen sollte sich auf solche beschränken, die für eine spätere Analyse interessant sind. Nebensächliche Funktions-

aufrufe wie „Datumsanzeige“, „Rückkehr zum Hauptmenü“ u. a. sollten ausgeblendet werden.

Neben Durchschnitt und Spitzen ist die Aufnahme des zeitlichen Verlaufs über typische Arbeitstage von Interesse. Hier zeigen sich zeitlich abhängige Engpässe, die unter Umständen zunächst einmal ohne großen technischen Aufwand organisatorisch entkoppelt werden können. Bei der Anzahl User, die hinter all diesen Ressourcen-Verbräuchen steht, ist zu differenzieren zwischen aktiven und lediglich über Sessions verbundenen.

2.2.2 Datenhaltungskonzepte

Ein weites Gebiet und damit großes Potenzial beim Anwendungstuning betrifft die Datenhaltung. Alle wesentlichen Computer besitzen auf die eine oder andere Weise Dateiverwaltungen als Teil des Betriebssystems oder als separates Werkzeug mit den Möglichkeiten, für Dateien logische Namen zu vergeben und ein entsprechendes physisches Speichermedium zu spezifizieren. Das System oder Werkzeug organisiert den physischen Platz und schreibt Daten hinein oder liest Daten heraus, je nach Anwenderbefehl.

Die weithin gebräuchliche Bezeichnung „Datenbasis“ bedeutet im weitesten Sinne eine Sammlung von Datensätzen, die untereinander mit maximaler Kohärenz verbunden, mit einer kalkulierten Redundanz gespeichert und auf eine Weise strukturiert sind, dass deren Nutzung einfach zu handhaben und eine Vielfalt von Abfragen durch viele Anwender für deren unterschiedliche Belange zu befriedigen sind (Abb. 2.12). Im Folgenden soll kurz auf die unterschiedlichen Datenhaltungssysteme eingegangen werden. Dabei bleibt es unvermeidlich, dass auch historisch gewachsene Strukturen, die eventuell nicht dem neuesten Stand des File Managements entsprechen, mit behandelt werden (Abb. 2.13 und 2.14). Grund dafür ist, dass auch heute noch in vielen Organisationen traditionelle Datenhaltung betrieben wird, die einerseits noch nicht durch moderne Modelle abgelöst worden ist, andererseits aber häufig Ursache für die tatsächlichen Performance-Probleme darstellt.

2.2.2.1 Technische Voraussetzungen

Ein allgemeines Datenmanagement-System erscheint dem Anwender als eine Software-Schnittstelle, die ihn sowohl vom Betriebssystem als auch von der externen Speicher-Hardware trennt, bezogen auf jeden Zugriff zu einer zentral kontrollierten und integrierten Datensammlung, die geteilt wird zwischen mehreren Anwendern und „Datenbasis“ genannt wird. Das System sieht Werkzeuge vor, um die physische Struktur der Datenbasis und die logischen Verbindungen in ihr zu definieren, um Daten zu laden und zu modifizieren, um diese Daten gegen zufällige Beschädigung oder unerlaubten Zugriff zu schützen und um für effiziente Datenabfragen zu sorgen. Ein Datenmanagement-System ist „allgemein“ zu nennen, wenn es eine Anwender-orientierte Kommandosprache für all diese verschiedenen Funktionalitäten

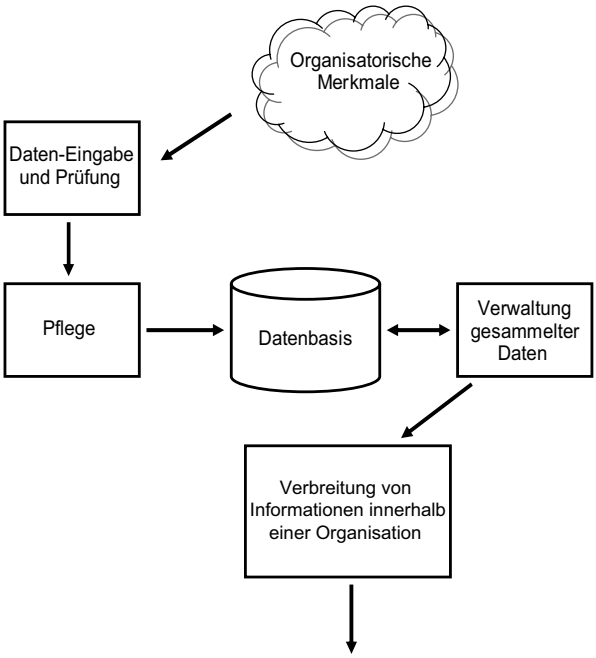

Abb. 2.12 Datenbasis innerhalb einer Organisation

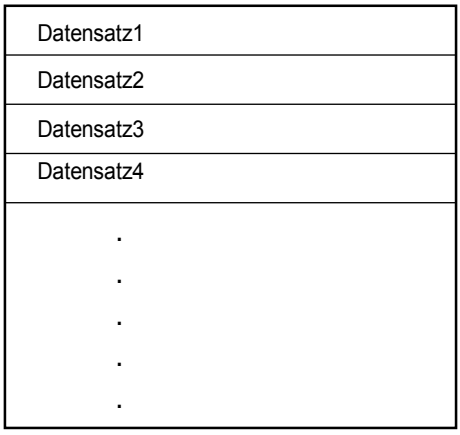

Abb. 2.13 Sequentielle Datei

mit sich bringt, die anwendbar ist auf jede neue Datenbasis, unabhängig von ihrer internen Organisation, um somit die Notwendigkeit abzuschaffen, jedes Mal neue Datenverwaltungsprogramme für jede neue Datenbasis zu schreiben.

Aber selbst in solch einer komfortablen Umgebung bleiben zwei ungelöste Probleme:

- die Optimierung der Daten-Abbildungen
- die bestmögliche Nutzung fortschrittlicher Datenbasis-Verwaltungsfunktionen.

Schon 1971 wurden Standards für DBMS (Data Base Management System) durch die CODASYL Data Base Task Group entwickelt – und zwar damals für COBOL

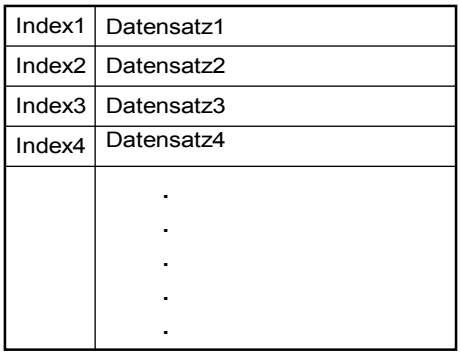

Abb. 2.14 Index-sequentielle Datei (ISAM)

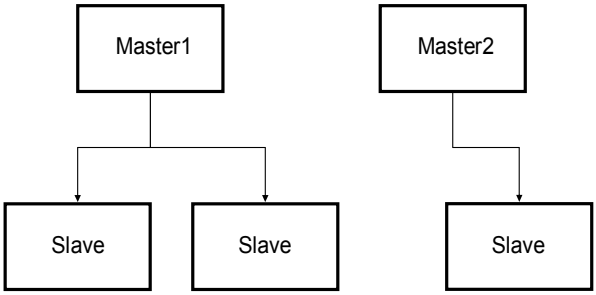

Abb. 2.15 Klassisches DBMS

als Haupt-Host-Sprache. Vorgeschlagen wurde eine Netzwerk-Struktur, um Relationen zwischen Datensätzen innerhalb der Datenbasis zu modellieren (Abb. 2.15). Hierarchische Relationen wurden als einfache Sonderfälle einer Verkettung von Netzwerk-Dateien abgehandelt.

Letztere verbreitete logische Struktur findet sich in der Vernetzung von Datensätzen, in der Felder innerhalb eines Satzes hierarchisch oder in Baumstruktur organisiert sind. Im Allgemeinen können die meisten Systeme eine Multi-Niveau-Hierarchie abbilden, in denen Unterfelder selbst wieder aus Unterfeldern bestehen können und so fort.

Reine logische Netzwerke basieren auf dem Konzept untereinander verbundener Dateien, wobei jede einen Besitzer- oder Master-Datensatz und einen oder mehrere Mitglieder- oder Slave-Datensätze zugeordnet haben kann. Die größere Leistungsfähigkeit von Netzwerken im Gegensatz zu hierarchischen Strukturen liegt in der Möglichkeit, eine Datensatzart mit vielen anderen zu assoziieren.

Ein typisches DBMS beinhaltet die folgenden Komponenten (Abb. 2.17):

- Datenbeschreibungs-Compiler
- Data Dictionary-Berichte
- Datenmanipulationssprache
- System-Module
- Utility-Programme.

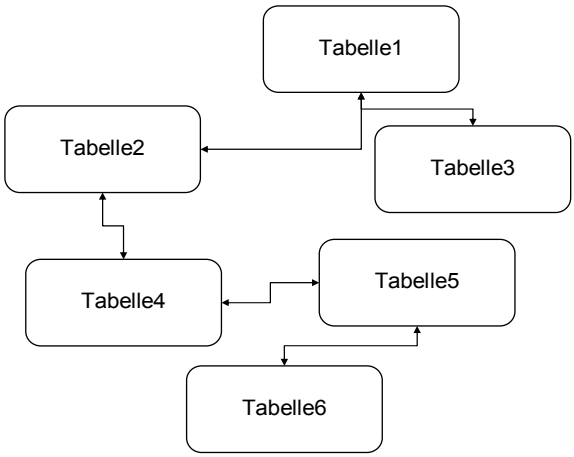

Abb. 2.16 Relationales Datenbank-Management-System (RDBMS)

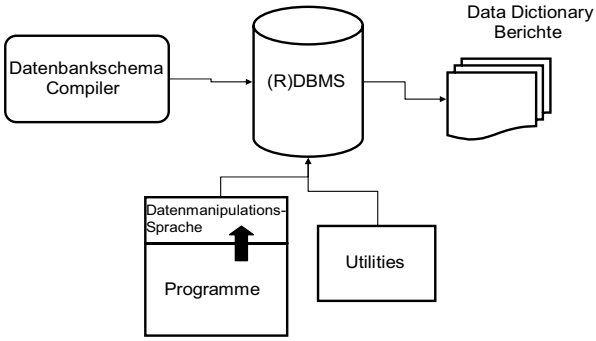

Abb. 2.17 DBMS-Werkzeugkiste

Schon bald nach seiner Einführung hatte das sogenannte relationale Modell großen Einfluss auf das Datenmanagement als solches (Abb. 2.16). Im Gegensatz zu den oben diskutierten Modellen kann das relationale Modell durch einfache, rigorose mathematische Konzepte definiert werden. Außerdem erlaubt seine unterliegende Struktur einfache Abfragen, Änderungen, Pflege und Neustrukturierung der Datenbasis.

Insgesamt ist das relationale Modell mehr anwenderorientiert. Es sieht eine Datenbasis als eine Sammlung von n-möglichen Relationen oder homogenen Tabellen vor, in denen jede Zeile einem Datensatz entspricht, der n Felder enthält, von denen keines mehrfach auftritt. Wenn man also Relationen definiert, können Konsistenz und Nicht-Redundanz garantiert werden, wenn man einer Gruppe formaler Regeln folgt.

Diese fünf grundlegenden Operationen in Relationsalgebra sind:

- Selektion
- Projektion
- Produkt

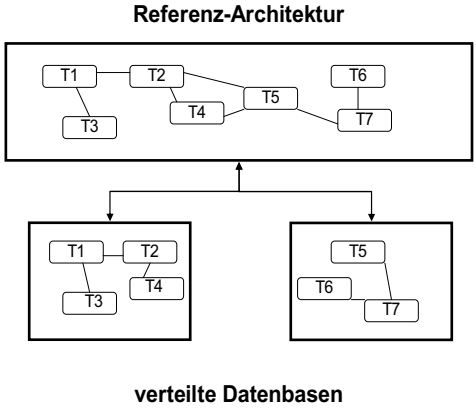

Abb. 2.18 Referenzarchitektur

- Vereinigung und
- Differenz.

Der nächste logische Schritt in Richtung Datenbasis-Technologie war die Entwicklung verteilter Datenbasen. Dem Anwender erscheint dieses Prinzip ziemlich einfach und ist ihm transparent: Anwendungen haben lokalen Charakter auf einer lokalen Plattform. Innerhalb eines Kommunikationsnetzes koordinieren Datenbasis-Funktionen Zugang zu verteilten Datenbasen und erzeugen dem Anwender den Anschein, als ob die Daten lokal verfügbar wären. Das Wissen über die tatsächliche Datenverteilung gehört dem DBMS allein. Änderungen der Datenverteilung ziehen keine Änderungen in Programmen nach sich.

Um das zu erreichen, verlässt sich das DBMS auf eine sogenannte Referenz-Architektur, die globale und lokale Schemata und die physische Datenzuordnung beinhaltet (Abb. 2.18).

2.2.2.2 Zugriffe

Unabhängig von den Datenhaltungssystemen im Einsatz bieten verschiedene Betriebssysteme Datenzugriffsmodalitäten an, die Einfluss auf die Gesamtperformance haben. Dazu gehören:

- gepufferter oder nicht-gepufferter Zugriff
- Zugriff mit oder ohne I/O-Wait

Beide Optionsmöglichkeiten erlauben unter Umständen die Weiterexekution von Programmen, ohne dass der I/O-Vorgang bereits abgeschlossen sein muss. Daneben spielen natürlich Tabellengrößen als solche eine Rolle sowie die gesamte Blockierphilosophie. Blocken sollte auf dem möglich niedrigsten Niveau (Item) geschehen, wenn möglich, keinesfalls auf Tabellenniveau, und möglichst auch nicht auf Datensatzniveau.

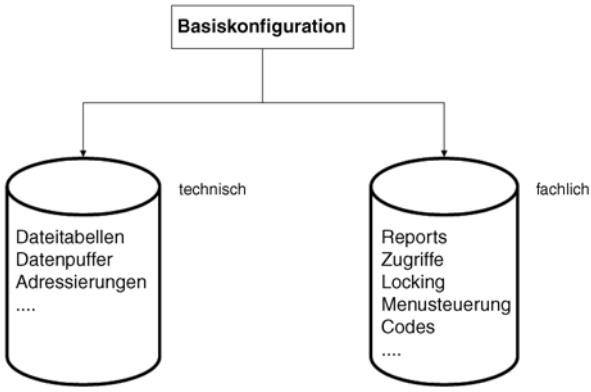

Abb. 2.19 Anwendungswelt

2.2.3 Anwendungsumgebung

Eine Anwendungsumgebung ist mehr als nur die für den User sichtbare GUI und die dazu gehörende Software und Datenbasis, die seine Transaktionen – inklusive Batch – unterstützt. Um das Verhalten der reinen Anwendung zu verstehen, muss man das Zusammenspiel zwischen den Systemkomponenten und den Anwendungen selbst betrachten. Es handelt sich also immer um eine Symbiose dieser beiden Aspekte. Um das Gesamtsystem zu beschreiben, müssen berücksichtigt werden:

- der Systemkern: das Betriebssystem mit seiner Verwaltung der angebotenen Systemressourcen
- das Job-Management
- das Datenmanagement auf Betriebssystemebene
- Ein-/Ausgabe-Steuerung
- sonstige externe Ressourcen (Speicher, Modems etc.).

Zu einer Anwendung gehören unter Umständen auch Subsysteme und Utilities wie Excel Sheets, Verzeichnisse und andere. Beim Standbau für eine Anwendung sind daneben noch andere Variablen zu berücksichtigen, wozu gehören:

- Anwendungs-spezifische Puffer und Adressräume
- Berechtigungen
- Prioritäten
- sonstige Betriebseinstellungen.

All diese Einstellungen haben Einfluss auf die Gesamt-Performance, so z. B. die Konfiguration von Puffern und Adressraum auf den Hauptspeicher. Insofern sind Kompromisse bei konkurrierenden Ressourcen-Anforderungen unausweichlich. Abbildung 2.19 zeigt noch einmal schematisch die Welt, in der sich Anwendungen bewegen:

Zu einer Produktivsetzung einer Anwendungsumgebung gehören ganz allgemein:

- das Software-Release
- die zugehörige Datenbank
- zusätzliche Utilities
- Dokumentation
- Schulung
- Berechtigungen.

http://www.springer.com/978-3-642-17189-5

Performance-Optimierung
Systeme, Anwendungen, Geschäftsprozesse
Osterhage, W.W.
2012, XII, 176 S. 97 Abb., Hardcover
ISBN: 978-3-642-17189-5