Systematische Performance-Analyse mit Oracle Solaris

Volker A. Brandt
Brandt & Brandt Computer GmbH
Am Wiesenpfad 6, 53340 Meckenheim

Ulrich Gräf
Oracle Deutschland B.V. und Co. KG
Riesstr. 25, D-80992 München

Schlüsselworte
Solaris, Linux, Unix, Performance, Systematik

Einleitung
Die Performance von Applikationen ist wichtig für den Betreiber eines Rechners. Der eigentliche Grund für den Betrieb des Rechners ist ja in der Regel die Applikation.
Nun kann es sein, dass der Betreiber oder die Nutzer der Applikation zu dem Schluss kommen, dass die Applikation zu langsam ist. Daraus begründet strebt man eine Steigerung der Leistung an.
Die Performance-Analyse liefert die Schwachpunkte und gibt gegebenenfalls auch Wege zur Steigerung der Leistung vor.
Dieser Vortrag zeigt die Performance-Analyse aus der Sicht des Betriebssystems anhand von Beispielen aus Solaris, und wie man die Defizite beheben kann.
Mit entsprechenden Tools ist die Vorgehensweise jedoch allgemein für Unix und Linux anwendbar. Sogar die Windows-Betriebssysteme von Microsoft nutzen die gleichen Mechanismen, so dass die Vorgehensweise anwendbar ist, jedoch andere Werkzeuge benötigt.
Es kann eine weitergehende Analyse der Performance in der Applikation notwendig werden (z.B. bei Datenbanken), wenn alle Hindernisse aus Betriebssystem-Sicht beseitigt sind. Die Behebung kann dann durch Neu-Konfiguration der Applikation (z.B: Anpassung der Queries) oder muss notfalls durch Neu-Programmierung erfolgen.

Ist eine Performance-Analyse notwendig?
Das ist die Frage, die zuerst beantwortet werden muss. Ein Tuning ohne wirklichen Grund verschlingt nur Kosten ohne eigentlichen Gewinn. Letztlich ist bei der Beantwortung dieser Frage nur die Wirkung der Applikation auf die Nutzer wichtig.
Eine reine System-Beobachtung reicht hier nicht aus, wie z.B. eine Aussage wie "bei der Ausgabe von `sar` steht der load immer über 35". Das zeigt einen internen Systemwert an. Egal wie der steht, wenn der User zufrieden ist, ist das völlig ausreichend.
Eine immer wieder vorkommende Anforderung ist: "Sie sind doch Performance-Spezialist, sehen Sie doch bitte mal auf unser System, ob alles OK ist.". So etwas kann man machen, aber solange die Applikation zufriedenstellend läuft, besteht hier keine Notwendigkeit, es werden nur Kosten generiert.

Was ist denn nun ein echter Grund, eine Performance-Analyse durchzuführen?
Wenn die Nutzung oder Beobachtung der Applikation ergibt:

- dass die Antwortzeiten von Requests zu langsam sind
- die Datenraten zu gering sind
- Laufzeiten von Batch-Jobs (Reports, Datenbank-Import, -Export, ...) zu lang sind.

Ein wichtiger Parameter bei der Performance-Analyse ist der Ziel-Wert, der erreicht werden muss. Erfolg ist definiert durch Erreichen eines Ziels. Umgekehrt wird daraus, dass bei einem unklar definierten Ziel der Erfolg niemals erreicht werden kann.
Anforderungen wie

- "Machen Sie die Maschine so schnell wie möglich!"
- "Die Maschine kann doch mehr, optimieren Sie!"

können daher niemals Erfolg haben, weil kein klares Ziel definiert ist.

Strategie vs. Taktik

Strategische Benchmarks wie TPC-C, TPC-D, ..., SPECint, SPECfloat, ... erzeugen Zahlen zum Veröffentlichen. Das sind in der Regel gute Zahlen - weil schlechte Zahlen nicht veröffentlicht werden - und sie zeigen die prinzipielle Leistungsfähigkeit von Systemen. Bei strategischen Benchmarks wird sehr viel getunt und in der Regel nur ein Aspekt einer Applikation betrachtet: die höchste Performance. Die Einbindung in den Betrieb in Zusammenarbeit mit anderen Datenbanken oder Daten liefernden oder verbrauchenden Systemen ist bei strategischen Benchmarks nicht berücksichtigt. Diese Verbindungen werden jedoch im realen Rechenzentrum benötigt. Daher können die Leistungen von veröffentlichten strategischen Benchmarks niemals wirklich erreicht werden.

Ein taktischer Benchmark oder „Proof-of-Concept“ (PoC) ist sehr viel näher an der Produktion beim Kunden. In der Regel definiert der Kunde das, was getestet werden soll, und er hat die Möglichkeit, seine kritischen Bedingungen in den PoC einzubringen. Damit kann sichergestellt werden, dass die Applikation später im Betrieb die notwendige Leistung bringt.

Systematische Performance-Analyse

Die klassischen Bereiche, die zu überprüfen sind:

- CPU
- Memory
- Disk-I/O
- Netzwerk-I/O
- Systemlast in Verbindung mit der Applikation
- Applikations-Konfiguration oder -Programmierung

Man kann die entsprechenden Messungen parallel durchführen, allerdings sollte bei der Analyse diese Reihenfolge wenn irgend möglich eingehalten werden.

Wenn die CPU-Leistung erschöpft ist, dann kann durch keine andere Maßnahme die Leistung erhöht werden, da schlicht die anderen Komponenten des Systems nicht beauftragt werden können. Als erstes muss daher die Erschöpfung der CPU-Leistung geprüft werden.
Erst danach sollte man nach der Auslastung des Speichers sehen. Sofern nicht genügend Hauptspeicher zur Verfügung steht oder der Zugriff darauf zu langsam ist, leidet die Performance der Applikation.
Erst danach ist man sicher, dass die Ein- und Ausgabe nicht durch CPU und Memory behindert wird, und man kann unbeeinflusst Disk- und Netzwerk-IO untersuchen.
Es gibt Fälle, bei denen weder CPU, Memory, Disk noch Netzwerk die Applikation verlangsamen, dann ist in der Regel die Applikation selbst der Verursacher. Einige Dinge können vom Betriebssystem her beobachtet werden. Ansonsten bleibt nur der Blick in die Applikation.

Wichtig ist auch, dass bei wiederholter Performance-Analyse die Reihenfolge eingehalten wird. Nicht selten ist nach einer Verbesserung z.B. im Disk-Bereich der limitierende Faktor an eine andere Stelle gewandert (z..B. CPU).

Analyse der CPU-Last

Die wichtigsten Fälle, in denen das Limit bei der CPU liegt:

- 100 % CPU Auslastung
- ein Thread hält die Applikation auf
- ein zentraler Prozess der Applikation wird zu oft verdrängt
- zuviel Systemlast

Wenn im System alle CPUs zu 100% arbeiten (idle = 0% im `vmstat`) dann kann die Applikation nicht mehr schneller werden, weil keine CPU-Zyklen mehr für sie zur Verfügung stehen.
Abhilfe schaffen hier nur mehr CPUs oder schnellere CPUs.

Manche Applikationen nutzen die mögliche Parallelität nicht aus, sie sind *single-threaded*. Seit einiger Zeit (bei Solaris sind es 20 Jahre) können Prozesse mit mehreren Threads arbeiten, die auch vom Betriebssystem auf mehrere CPUs (Cores, Threads) verteilt werden können (*kernel threads*). Heutige Rechner haben häufig mehrere CPUs (Sockets), jeder von denen hat mehrere Cores, die wiederum mehrere Threads haben (z.b. beim SPARC T5-Prozessor bis zu 8 Sockets mit je 16 Cores, die je 8 Threads haben: 1024 parallele kernel threads). Die Anzahl der Threads, die eine Applikation nutzt, ist z.B. im `prstat`-Kommando zu sehen (Spalte `NLWP`). Wenn nur ein Teil der Prozessoren voll ausgelastet ist, lässt sich das mit `mpstat` ermitteln.
Die Behebung besteht darin, die Applikation mit mehr Threads zu konfigurieren, sofern möglich. Oder man installiert noch mehr Applikationen auf diesem Rechner, ggf. abgetrennt mit Virtualisierung (OVM, LDoms, Zonen, Virtualbox, VMWare, ...); dann steigt die Gesamt-Auslastung, aber nicht die Leistung der einzelnen Applikation.

Manchmal gibt es einen zentralen Prozess in einer Applikation, der die Verarbeitung steuert. Weil aber die Applikation intensiv arbeitet, kann es sein, dass der Prozess häufig die CPU verlassen muss.
Bei manchen Applikationen kann das vorkommen, dass der Listener-Prozess ein solcher Prozess ist. Dieser Prozess nimmt die Aufträge aus dem Netzwerk entgegen, wird dann verdrängt, während die Aufträge bearbeitet werden, und kommt erst wieder danach zum Zug. Resultat is ein schubweises Arbeiten der Applikation. Finden kann man solche Prozesse mit `ps` oder `top`, dort wechseln sie häufig die CPU, weil sie nicht immer der gleichen CPU zugeteilt werden.

Verifizieren des Verhaltens und Beheben des Problems geht mit Zuteilung einer eigenen CPU (mittels `psrset` oder `poolcfg/pooladm`) oder Zuteilung einer höheren festen Priorität (Schedulingklasse FX).

Wenn zuviel Systemlast vorliegt, liegt das in der Regel an der Applikation und kann nur durch Neu-Konfiguration oder Änderung der Programmierung behoben werden.

Analyse von Engpässen im Speicher

Die heutigen Rechnersysteme laufen in der Regel mit virtuellem Speicher. Das reicht von Mainframes über Unix, Linux, Windows ... bis zu den meisten Smartphones. Dabei wird der eingebaute Hauptspeicher mit Platz auf einem Massenspeicher (meist Platte) ergänzt und dieser wird bei Bedarf in den Hauptspeicher eingelagert. Anfangs (1970 bis ca. 1995 mit Batchbetrieb) war das akzeptiert, aber heutzutage wird es nur noch bei Desktops toleriert, ein Rechner im Rechenzentrum soll kein paging (swapping) betreiben, weil sonst die Performance der interaktiven Applikationen zu schlecht wird.

Daher ist es wichtig, dass der Hauptspeicher nach dem Speicherverbrauch der Applikationen konfiguriert wird. Viele Datenbanken, aber auch andere Applikationen nutzen gemeinsamen Speicher (`shared memory`), um schnell zwischen Prozessen zu kommunizieren. Auf Solaris kann man den Speicher im Hauptspeicher „verdrahten“ (ISM = intimite shared memory), um zu garantieren, dass wichtige Speicherbereiche niemals auf den Sekundärspeicher ausgelagert werden. Dieser Speicher steht dann natürlich auch anderen Applikationen niemals zur Verfügung und kann zum Speicherengpass führen.

Um einen Speichernotstand festzustellen, kann man mit `vmstat` betrachten, wieviel realer und wieviel virtueller Speicher belegt ist. Die Menge des freien Speichers steht in der Spalte f`ree` (Angabe ist in Kilobytes). Die Menge des belegten virtuellen Speichers ist in der Spalte `swap`. Zur Beurteilung der Lage braucht man noch die Menge des Hauptspeichers, die man mit `prtconf` oder `prtdiag` ermitteln kann.
Es sollte mehr als 1/32 des Hauptspeichers frei sein, weil ein System mit virtuellem Speicher für viele Vorgänge (Programmstart, Disk-I/O, Netzwerk-I/O, ...) freie Seiten braucht.
In der Spalte `sr` (scan rate) vom Kommando `vmstat` steht, ob das System aktiv nach Seiten sucht und Seiten auslagert. Dieser Wert sollte dauerhaft 0 sein. Nur bei Programmstarts, speziell bei Start von Programmen mit viel shared Memory, darf der Scanner suchen, was aber nach Ende des Startups aufhören muss.
Wieviel Speicher ein Prozess benötigt, ist mit dem Kommando pmap ersichtlich.
Für Dauerbetrieb ist eine Planung des Speichers wichtig. Das OS braucht 1-4 GByte, große Systeme mit viel Peripherie noch mehr, das 1/32 wird für den Freespace gebraucht, die Applikationen brauchen ihren Platz. Wenn Daten von Filesystemen gelesen werden, muss der Cache der Filesysteme ebenfalls berücksichtigt und/oder konfiguriert werden (UFS: segmap, ZFS: ARC). Manche Installationen haben dynamischen Speicherbedarf, weil manche Applikationen (z.B: Backup, Datentransfer, ...) nur zeitweise da sind. Dafür ist ebenfalls Platz vorzusehen.
Reduzierte Performance wegen Paging wird heutzutage nicht mehr akzeptiert, daher sollte mit diesen Größen der Hauptspeicher geplant werden.
Der Swap-Space ist noch für temporäre Dateien (/tmp) da und sollte ausreichend konfiguriert werden, weil sonst die temporären Dateien Hauptspeicher belegen. Er sollte auch noch Reserve für Notfälle enthalten (bei denen paging akzeptabel ist).

Die Werte page-in und page-out im `vmstat` dagegen sagen gar nichts aus, weil moderne Systeme mit virtuellem Speicher auch die ganz normale I/O mittels des Paging-Mechanismus durchführen.

TLB-Miss: Ein anderer Engpass bei der Benutzung des Speichers von der CPU entsteht, wenn die Applikation zuviele Speicherseiten gleichzeitig benutzt. Zum Lesen von Daten aus dem Hauptspeicher muss die CPU die im Programm verwendete virtuelle Adresse in eine reale Adresse des Hauptspeichers umwandeln. Dies geschieht unter Nutzung mehrere Tabellen und dauert einige Zeit (~ 100 Nanosekunden = 300 Befehls-Zyklen). Um das zu beschleunigen, gibt es in der Hardware der heutigen CPUs einen speziellen Cache, der die Umsetzung ohne Verzögerung beim Zugriff durchführen kann, wenn die Seite kürzlich schon mal in Benutzung war: der TLB (*translation lookaside buffer*). Durch die Anforderung an Geschwindigkeit ist der TLB aber in der Größe limitiert, in der Regel hat er nur 64 Einträge. Das heißt, wenn der Thread beim Ausführen der Applikation nur die Seiten benutzt, deren Umsetzung schon im TLB steht, ist der Speicherzugriff schnell. Benutzt die Applikation aber mehr Seiten parallel, dann müssen ständig Adressen umgesetzt werden, und die Applikation wird langsamer.
Diese Zeit wird sogar der CPU-Zeit im User-Bereich zugeordnet, so dass häufige Adress-Umsetzungen nicht im `vmstat` sichtbar sind. Mit den Kommandos `cputrack` und `cpuinfo` kann aus den CPUs ausgelesen werden, ob häufig sogenannte TLB-Misses auftreten und die Applikation beim Speicherzugriff verlangsamen.
Abhilfe beim TLB-Miss kann durch Nutzung größerer Speicherseiten erfolgen (Default: 8k bei SPARC, 4k bei x86). Bei shared memory-Segmenten wird schon lange (Solaris 2.6) automatisch eine größere Segmentgröße (4 MB) verwendet. Seit Solaris 9 kann man bei der Anforderung im Programm die gewünschte Seitengröße angeben und auch von außerhalb ohne Änderung des Programms die Standard-Seitengröße für verschiedene Bereiche vorgeben (Kommando `ppgsz`).

Analyse der Disk-I/O

Die Nutzung der Platten läßt sich am einfachsten mit dem Kommando `iostat` (`iostat -xzn 5`) untersuchen. Die Plattennamen werden im Klartext auf der rechten Seite angezeigt. Links stehen die Anzahl der Lese- und Schreib-Vorgänge und die entsprechenden Datenraten.
Die Überlastung ist definitiv da, wenn in der Spalte `wait` dauerhaft ein Wert >0 angezeigt wird. Dann nimmt die betreffende Platte keine Aufträge mehr an, und diese müssen im Betriebssystem warten.
Ansonsten steht in der Spalte `actv`, wieviele Aufträge auf dieser Platte im Mittel gleichzeitig arbeiten. Für beste Antwortzeiten sollte dieser Wert nicht größer als 1 werden, weil sonst der zweite Auftrag auf den ersten warten muss, und die Antwortzeit schlechter wird.
Für hohen Durchsatz ist es akzeptabel, dass mehrere Aufträge auf der Platte arbeiten, die Antwortzeiten werden zwar schlechter, aber der Gesamtdurchsatz steigt, weil die Platte intern optimieren kann (selten bei SATA, sicher bei SAS, FC, SCSI).
Die Antwortzeit ist in der Spalte `asvc_t` sichtbar.

Bei Random-Zugriff wird die mittlere Antwortzeit durch die Umdrehungszahl der Platte bestimmt. Eine 3 TB-SATA Platte z.B. hat in der Regel 7200 RPM, was 120 Umdrehungen pro Sekunde sind. Eine Umdrehung dauert also etwa 8 Millisekunden. Um an eine beliebige andere Stelle zu kommen, muss sich der Kopf bewegen (1-2 ms), und die Platte muss sich an die richtige Stelle drehen (0-8 ms, im Mittel 4 ms), dann muss sich der Kopf noch auf der Spur stabilisieren; in Summe ist das dann eine mittlere Zugriffszeit von ca. 8.5 ms (Datenblatt).

Zur Antwort an den Rechner benötigt man dann noch die Übertragung der Daten von der Platte in den Puffer der Platte und vom Puffer über den Anschluss (SATA, FC) zum Rechner.

Man kann daher bei einer 7200-RPM-Platte von ca. 100 Random-I/Os pro Sekunde ausgehen, das sind ca. 10 ms Antwortzeit.

Steigt die Antwortzeit über diesen Wert, dann liegen mehrere Aufträge parallel vor, die nacheinander bearbeitet werden; der Durchsatz steigt, die Antwortzeit wird schlechter.
Parallele Aufträge können auch schubweise erfolgen, so dass die Gesamtlast gar nicht gross ist, aber trotzdem die Antwortzeit schlecht.

Häufig wird die Spalte `busy%` fehlinterpretiert. Wenn die Platte 100% busy ist, dann liegt noch keine Überlastung vor. Platten können mehrere Aufträge parallel bearbeiten und der Durchsatz steigt dann. Man weiß nur, dass bei weniger als 100% wahrscheinlich die Aufträge einzeln kommen und wahrscheinlich die Antwortzeit gut ist.
Bei mehr als 100% kann aber die Antwortzeit auch noch akzeptabel sein. Die Leistung kann sogar sehr gut sein, wenn die Platte eine LUN von einem großen Storage-Subsystem mit RAID und batteriegepuffertem Cache ist, da mehrere reale Platten in der LUN sind und viele unabhängige I/Os ausführen können
In allen diesen Fällen ist aber die Spalte `asvc_t` mit der Antwortzeit ein viel besserer Indikator für die gelieferte Leistung als die Spalte `busy%`, die nur anzeigt, welchen Anteil der Zeit die Platte mindestens einen Auftrag hat.

Bei Konsolidierungssystemen, wo viele Applikationen aktiv sind, kann mit DTrace einfach ermittelt werden, welche Applikationen viele oder wenige IOs durchführen und wie die Antwortzeiten sind.

Schlechte Antwortzeiten bei Platten können durch Einsatz von mehr physischen Platten (striping), Auflösen von RAID5 (durch Mirror) oder durch Einsatz von Plattensubsystemen mit Cache (Oracle Sun ZFS Storage Appliance, Oracle Pillar Axxiom, Oracle Sun 2540 Array, oder anderer Hersteller) behoben werden.
Ein neuer Weg, die Antwortzeit zu verbessern, ist der Einsatz von SSDs , darauf können die wichtigen Daten vollständig abgelegt werden. Ein interessanter Weg die mit geringem Aufwand zum guten Ergebnis zu kommen ist mit dem automatischen Einsatz von SSDs für den wichtigen Teil der Daten zum Beispiel wie im ZFS Filesystem, den ZFS Storage Appliances oder den Exadata Systemen.
Durchsatzraten lassen sich im gewissen Masse steigern, indem die Software genug parallele Streams nutzt, die Antwortzeiten werden aber schlechter. Danach bleibt auch nur der Einsatz von mehr Platten.

Messung von Netzwerk-I/O

Die Netzwerk-Anschlüsse haben in der Regel eine gute Datenrate und stellen keine Flaschenhälse dar. Für verschiedene Anwendungen gibt es vorgeschlagene Einstellungen, und auch mit den Defaults kommt man in der Regel weit.

Meist kann man ein Limit beim Netzwerk mit `netstat -I` sehen bzw. mit `kstat` die Datenrate auf dem Netzwerk ermitteln.
Ein einfaches Mittel, um festzustellen, dass das Netwerk das Limit ist: Eine Verbindung von einem anderen Rechner ist langsam oder schlecht, vom gleichen Rechner aus ist es schnell genug.

Die Behebung von Engpässen ist durch schnellere Netzwerk-Interfaces (10Gbit, Infiniband) möglich. Gegebenenfalls hilft auch Zusammenschalten von Netzwerk-Anschlüssen (Trunking, Aggregation, Etherchannel, ...) wobei zu beachten ist, dass eine 1:n Kommunikation beschleunigt werden kann, eine Beschleunigung einer Komunikation zwischen zwei Rechnern aber nur unter bestimmten Bedingungen funktioniert.

Applikation und Systemlast

Wenn mit `vmstat` eine hohe Systemlast gemessen wird, dann ist dies meist von der Applikaton verursacht (sie wird ja auch bei `ps` bei dem Prozess der Applikation angezeigt).

Einige der auftretenden Fälle lassen sich von außerhalb der Applikation analysieren:

Spinning Mutex: Das Betriebssystem synchronisiert viele Dinge durch Locks. Der Thread, der das Lock hält, darf den durch das Lock geschützten Datenbereich benutzen, alle anderen nicht.
Zur Implementierung gibt es zwei Wege. Implizit wird immer angenommen, der andere Thread wird solange suspendiert, bis der eine Thread das Lock wieder freigibt (event-gesteuertes Lock). Es gibt jedoch Fälle, bei denen die geschützten Aktionen nur wenige Maschineninstruktionen sind (z.B. Bearbeiten eines Eintrags in einer Tabelle) und die Suspendierung des Prozesses mit 1000 - 10000 Maschineninstruktionen viel zu aufwendig wäre. Hier wird ein sogenanntes *spinning lock* oder *spinning mutex* verwendet. Statt zu suspendieren, wartet man einfach aktiv in einer Schleife. Durch geschickte Wahl der Anzahl der Locks und Segmentierung des geschützten Bereiches kann man die gleiche Sicherheit wie mit Suspendierung bekommen und muss nur sehr selten in der Schleife warten.

Die *spinning mutex*-Synchronisation ist auch von Benutzerprogrammen aus einsetzbar, kann aber bei ungeschickter Nutzung zu einer langsamen Applikation führen.
Summarisch kann man beim Kommando `mpstat` sehen, wieviele spinning mutex jede CPU durchführt (Spalte `smtx`, viel heisst > 20000 pro CPU). Wenn mehrere Applikationen aktiv sind, kann man mit `lockstat/plockstat` oder `dtrace` die spinning mutex der Applikation / dem Prozess zuordnen. Abhilfe schaffen kann man mit weniger Parallelität der Applikation, schnelleren CPUs oder einer Neu-Konfiguration der Applikation (z.B: weniger Synchronisationen und größere Puffer). Als letztes Mittel kann sogar eine Änderung der Programmierung der Applikation notwendig sein.

Systemcalls: Applikationen fordern Betriebssystemdienste mit sogenannten Systemcalls an (z.B. open(), read(), write(), ...).
Ein Beispiel für eine ungünstige Nutzung von Systemcalls ist z.B. die byteweise Ausgabe direkt auf die Platte, wobei jede Ausgabe einen Systemcall auslöst und viele Aktionen des Betriebssystem für jedes einzelne Byte erfordert.
Ein anderes Beispiel ist eine Applikationsarchitektur, in der ein Prozess die Daten hält und viele andere Prozesse per Semaphoren-Synchronisation den Datenhaltungs-Prozess benutzen. Wenn die Datenmengen pro Synchronisation zu klein sind und/oder zuviele Prozesse bei der Semaphore in Konflikt kommen, dann muss die Applikation angepasst (oder umkonfiguriert) werden.

Die Systemcalls sind ebenfalls im `mpstat` sichtbar, Spalte `syscall`. Die Zuordnung zu Prozessen kann mit DTrace oder mit `truss -c` bestimmt werden.

Prozessor-Crosscalls: Die heutigen Systeme haben viele Threads, auf denen häufig Applikationen mit verteiltem Speicher laufen. Das sind entweder Prozesse mit mehreren Threads, die den gemeinsamen Speicher des Prozesses nutzen, oder unabhängige Prozesse mit Shared-Memory-Segmenten. Das Rechnersystem muss in diesem Fall die Cache-Kohärenz garantieren.

Am häufigsten kommt vor, daß ein Thread Daten in den Hauptspeicher geschrieben hat, die Daten aber zunächst nur im Cache stehen, und die Änderung den Hauptspeicher noch nicht erreicht hat. Dann muss das Rechnersystem sicherstellen, dass andere Threads, die den gleichen Speicher benutzen, die richtigen Daten erhalten.
Die Implementierung erfolgt zum Beispiel so, dass jeder Prozessor weiß, ob Daten shared sind oder nicht und ob die Daten von einem anderen Prozessor verändert wurden (z.B. mit einem sogenannten *snooping bus*).
Wenn dann Prozessor A Daten lesen soll und weiss, daß die Daten wurden von Prozessor B verändert, aber noch nicht im Hauptspeicher sind, dann verbindet sich Prozessor A direkt mit dem Prozessor B und lässt sich die Daten aus dem Cache von Prozessor B liefern. Dies nennt man einen Prozessor-Crosscall.

Früher wurden dann das Schreiben aller Daten vom Cache in den Speicher erzwungen. Das verringert die Leistung der Applikation schon bei geringer Parallelität (ab 3 Threads) wesentlich.

Applikationsarchitektur: Andere Leistungsverbesserungen in der Applikation lassen sich in der Regel nur über Änderungen der Applikation erzielen. Hilfsmittel dazu sind truss, dtrace und natürlich der Sourcecode der Applikation.

Zusammenfassung

Die hier vorgestellte systematische Performance-Analyse reicht in 90% der Fälle, aus um die Ursache für Performance-Probleme zu finden. Abhilfe kann in einigen Fällen durch neue Hardware (CPU, Memory, Netzwerk) oder Umkonfiguration (Disk, Netzwerk) geschaffen werden.
Ungünstige Konstellationen von System und Applikation können teilweise durch Umkonfiguration, aber häufig auch nur durch Anpassung der Applikation selbst behoben werden.

Kontaktadresse:
Volker A. Brandt
Brandt & Brandt Computer GmbH
Am Wiesenpfad 6
D-53340 Meckenheim

Telefon: +49 (0) 2225 946665
Fax: +49 (0) 2225 946666
E-Mail vab@bb-c.de
Internet: http://www.bb-c.de/

Kontaktadresse:
Ulrich Gräf
Oracle Deutschland B.V. & Co. KG
Robert-Bosch Str. 5
D-63303 Dreieich

Telefon: +49 (0) 6103 397 390
E-Mail ulrich.graef@oracle.de
Internet: http://blogs.oracle.com/solarium http://blogs.oracle.com/blug