Proceedings of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005). Atlanta, GA, September 25-27, 2005.

Continuous resource monitoring for self-predicting DBMS

Dushyanth Narayanan
Microsoft Research
Cambridge, UK
dnarayan@microsoft.com

Eno Thereska
Carnegie Mellon University
Pittsburgh, PA
enothereska@cmu.edu

Anastassia Ailamaki
Carnegie Mellon University
Pittsburgh, PA
natassa@cmu.edu

Abstract

Administration tasks increasingly dominate the total cost of ownership of database management systems. A key task, and a very difficult one for an administrator, is to justify upgrades of CPU, memory and storage resources with quantitative predictions of the expected improvement in workload performance. Current database systems are not designed with such prediction in mind and hence offer only limited help to the administrator. This paper proposes changes to database system design that enable a Resource Advisor *to answer "what-if" questions about resource upgrades. A prototype Resource Advisor built to work with a commercial DBMS shows the efficacy of our approach in predicting the effect of upgrading a key resource — buffer pool size — on OLTP workloads in a highly concurrent system.*

1. Introduction

Administering database management systems (DBMS) is a complex and increasingly expensive task, and there is a pressing need for greater automation in this area [7, 13]. A key aspect of DBMS administration is *resource provisioning*: given a hardware budget, an administrator must decide whether and in what proportion to invest in faster processors, additional memory, or larger and faster disks. DBMS running transactional workloads serve as back ends to a variety of enterprises such as e-commerce, banking, and travel reservation systems, essentially determining the application's response time [19]. Accurate database resource provisioning is thus vital to ensuring quality of service in these enterprises.

Such enterprises typically hire human experts who use experience and rules of thumb [12] to decide whether acquiring more resources will improve performance. The cost of hiring experts for resource provisioning decisions is high for large enterprises and practically prohibitive for the large number of small businesses using DBMS as back ends. Even experts find it difficult to *quantify* the expected benefit of a resource upgrade, an especially challenging task for highly concurrent OLTP (On-Line Transaction Processing) workloads whose behavior at any point in time is the combined effect of many concurrent transactions. The net effect is that DBMS are often over-provisioned.

In this paper we argue that the DBMS itself should provide automated answers to "what-if" questions about its resources by predicting the effect of proposed upgrades on both aggregate and per-transaction performance. To gain insights on the system changes required for such self-predictability, we have build a prototype *Resource Advisor* for a recent unreleased version of Microsoft's SQL Server DBMS. The Resource Advisor answers what-if questions such as "How would performance be affected if I doubled the amount of memory on this server?" with *quantitative* answers such as "throughput will increase by 40%, the response time of 'new order' type transactions will decrease by 80%, and the CPU will become the new bottleneck." Although many aspects of database performance have been individually studied in great detail, we believe this is the first attempt to automatically answer such high-level questions through an architecture that integrates live system tracing with hardware resource models and performance prediction.

One resource of great interest for performance prediction is the main memory buffer pool used for caching data retrieved from disk, often an important performance limiting factor [5, 9, 11]. Its effect on performance is workload-specific, non-linear, and hard to predict using rules of thumb. Although the Resource Advisor is designed to answer "what-if" questions about CPU and storage as well, our prototype implementation and evaluation focus on buffer pool size as the variable resource.

The contributions of this paper are as follows:

- We show the feasibility of answering "what-if" questions through a prototype implementation and evaluation of a Resource Advisor for a commercial DBMS.
- We present a modular architecture for the Resource Advisor, and identify the key components required for effective self-prediction: low-level instrumentation, end-to-end transaction tracing, and parametrized models of hardware resources.
- We demonstrate the additional benefits of our end-to-end tracing technique in visualizing and understanding system performance.

Our design and implementation are validated through detailed experiments showing that the Resource Advisor accurately predicts changes in OLTP workload performance across large changes in available main memory resources. When available memory is doubled, for instance, the Resource Advisor predicts the resulting throughput to within 7% or better. The Resource Advisor also correctly tracks the trend in transaction response time across a range of memory sizes spanning more than an order of magnitude.

2. Motivation and design

Large commercial databases are complex systems that depend on several physical resources such as the back end storage system, volatile main memory and CPUs. A database administrator (DBA) must decide on a good initial configuration of these resources, and then continuously monitor the system for new bottlenecks and changes in workload. There are two nightmare scenarios that every DBA faces. First, when clients complain that their workload performance does not meet service-level agreements, she needs to pinpoint the source of the problem. Second, a fixed budget is allocated to buying new hardware during periodic system upgrades. Which resources should the DBA upgrade and how can she quantify the effect on workload performance? From talking to administrators of real database systems, it is clear that they do not have the right tools to handle these scenarios.

The most common solution — over-provisioning all the resources that might impact performance — is wasteful and can be prohibitively expensive. A second approach is to monitor performance using the aggregate counters provided by most commercial DBMS [14, 21, 22]. Such per-resource counters provide a narrow view of the system and do not identify the global bottlenecks. (If the observed disk queues are long, should we buy more memory or faster disks?) Additionally, aggregate statistics do not offer any insights into response time, since they do not distinguish between background and foreground ("critical-path") resource usage. Finally, to determine the effect of a change in the available resources the DBA must still constructively interpret the performance implications of 400+ counters.

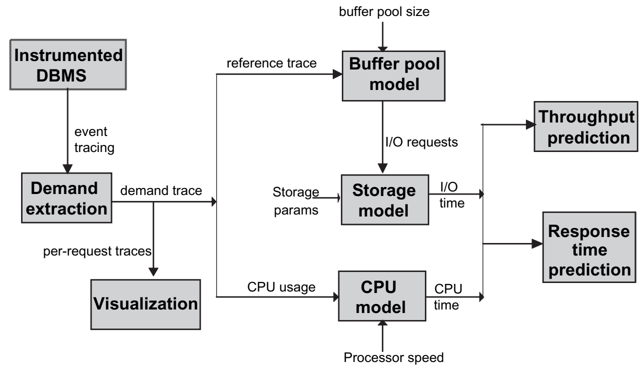


Figure 1. Resource Advisor components.

The key to optimal price/performance is the ability to answer "what-if" questions: given a system with some resources, to predict its performance with a different set of resources. The Resource Advisor's goal is to automatically answer such questions with *point predictions* ("what will happen to response time if I double the current memory?") as well as *trend forecasting* ("what is the shape of the memory-response time curve?"). A second goal is to provide detailed information about current system performance, allowing the DBA to query and visualize any aspect of performance or resource usage at any level of detail.

We expect that the Resource Advisor would be part of the DBMS software at the server site, continuously collecting trace information and maintaining a summary of workload resource demand based on recent behavior. At any point, the DBA could propose a resource upgrade and receive a quantitative prediction of the expected performance.

2.1. Design principles

Our design incorporates several insights we have had into enabling a DBMS to identify resource bottlenecks and predict the impact of upgrades. First, DBAs are rarely able to precisely characterize or model the application workload, which in any case will change over time. This means that resource advice should be based on *continuous monitoring* of a live system running a real application workload.

Second, OLTP performance in particular depends on the interaction between multiple DB components concurrently executing transactions of different types. Thus it is insufficient to track aggregate resource utilization statistics: we must *trace* system behavior in sufficient detail to compute the resources used by each transaction, and the order in which they were used.

Third, it is essential to separate *demand* from *service*. The former refers to the resource demands placed by the workload on the system, independent of the underlying

	Event Type	Arguments	Description
Control Flow	*StartRequest*		SQL transaction begins
	EndRequest		SQL transaction ends
	EnterStoredProc	*procname*	Stored procedure invocation
	ExitStoredProc	*procname*	Stored procedure completion
CPU scheduling	*SuspendTask*	*taskid*	Suspend user-level thread
	ResumeTask	*taskid*	Resumes user-level thread
	Thread/CSwitchIn	*cpuid, systid*	Schedule kernel thread
	Thread/CSwitchOut	*cpuid, systid*	Deschedule kernel thread
Buffer pool activity	*BufferGet*	*pageid*	Get reference to a buffer page (blocking)
	BufferAge	*pageid*	Reduce the "heat" of a page
	BufferTouch	*pageid*	Increase the "heat" of a page
	BufferDirty	*pageid*	Mark a page as dirty
	BufferReadAhead	*startpage, numpages*	Prefetch pages (non-blocking)
	BufferEvict	*pageid*	Evict a page to the free pool
	BufferNew	*pageid*	Allocate a new page from the free pool
Disk I/O	*DiskIO*	*startpage, numpages*	Asynchronously read/write pages
	DiskIOComplete	*startpage, numpages*	Signal read/write completion

Table 1. Events used by the Resource Advisor

hardware. The latter refers to the way these resource demands are scheduled by the available hardware resources. To answer "what-if" questions about resource changes, we must distinguish workload characteristics from hardware-dependent measurements.

Finally, we advocate that each individual resource manager (CPU scheduler, buffer manager, I/O scheduler) in the system be *self-predicting*, with the ability to answer hypothetical questions about its behavior under different resource regimes. The designers of these components are in the best position to incorporate predictive models that correctly reflect their scheduling algorithms, eviction policies, etc. Since current DBMS lack such models, our prototype Resource Advisor includes simple CPU, buffer pool, and disk models: our aim here is to validate the overall framework rather than to develop new modeling techniques.

2.2. Prototype architecture

Based on the above design principles, we have built a prototype Resource Advisor for SQL Server: Figure 1 shows its high-level architecture. Event traces are generated by the *instrumented DBMS*, and can be consumed online or written to disk for offline analysis. *Demand extraction* then separates out the hardware-independent aspects of the resource demand. This resource demand can then be *visualized* in various ways, and is also fed into workload-independent, parametric *resource models* which predict throughput and response time for hypothetical resource configurations.

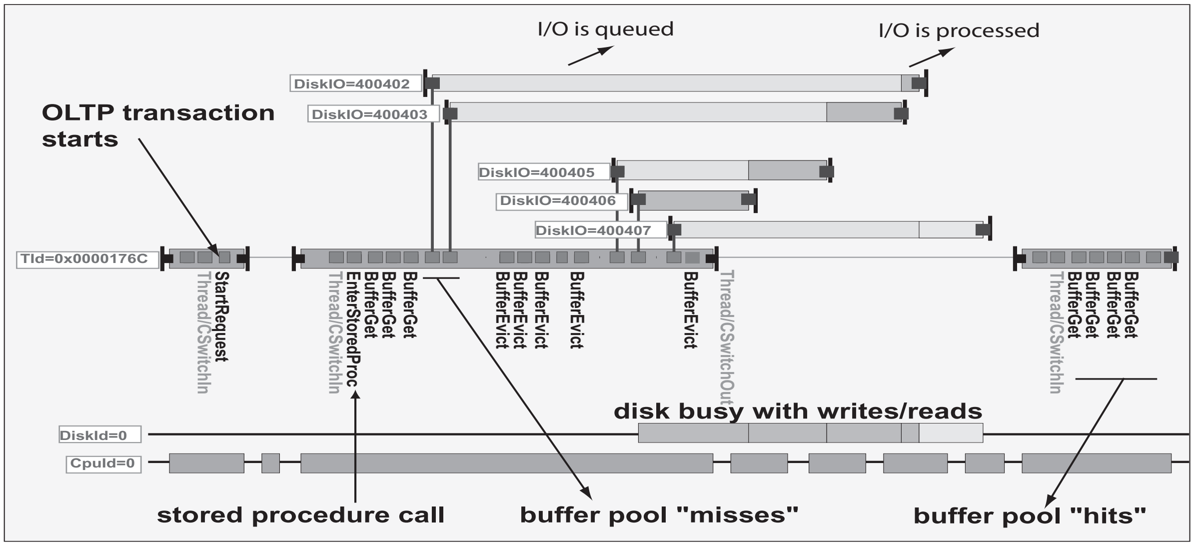
3. Resource Monitoring

This section describes the instrumentation required in a self-predicting DBMS to provide fine-grained, end-to-end traces of transaction resource demands; the extraction of aggregate and per-transaction demand traces; and their use in performance visualization. Section 4 describes their use in answering "what-if" questions.

3.1. Instrumentation

We have instrumented a private copy of the SQL Server source code to track the use of CPU, memory, and I/O resources. Each instrumentation point generates an *event* with associated parameters related to resource usage; these events are processed in time order by the Resource Advisor components to generate performance predictions. Events are inserted in the DBMS source code as calls to C functions, whose implementations are automatically generated from a high-level definition of the interface between the DBMS and the Resource Advisor. Each event is automatically annotated with the user and kernel thread IDs and then posted through Event Tracing for Windows (ETW) [20], a low-overhead tracing infrastructure in Windows Server 2003. ETW timestamps and orders the events using an accurate, high-resolution timer such as the processor cycle counter, and flushes event buffers in the background.

Currently we trace events relating to transaction control flow, buffer pool activity, disk I/O, and thread scheduling (Table 1). Although the exact interface and instrumenta-



The figure shows a timeline view of the resource demands and performance of a single OLTP transaction, extracted from a highly concurrent workload. The view was automatically generated by the Resource Advisor, and only the annotations in large bold font were added by hand. Due to space constraints, we only show an initial portion of the transaction. On the bottom three timelines, we can see when the transaction was being executed by a system thread, when the I/O system was busy processing this transaction's requests, and when the CPU was busy. The "thread" timeline also shows events occurring in the transaction execution path. Some of these events correspond to Disk I/O requests, which are shown in the top 5 timelines.

Figure 2. Timeline view of a single transaction from an OLTP workload

tion points will depend on the particular DBMS, this list of events represents the information that any DBMS will need to monitor for effective self-prediction. We are extending the instrumentation and modeling to include locking activity, which can have a significant impact on response time.

3.2. Demand trace extraction

A live system trace is the combined effect of workload demand and resource availability. From it, the Resource Advisor extracts a *demand trace* that represents workload behavior in a hardware-independent way. This is then used to represent workload behavior when modeling the effect of changing the hardware resources. The demand trace includes a *buffer reference trace*, which contains the resource-independent aspects of buffer pool activity: demand accesses, readaheads, buffer touches, buffer dirties, and new page creations. It does not include buffer evictions or I/O events, which depend on the buffer pool size. The demand trace also includes the CPU cycles used in executing workload transactions, computed by tracking the active threads within the DBMS using the scheduler context switch events.

The demand trace contains the interleaved demands of many concurrently executing transactions as well as background tasks such as buffer pool management. From it the Resource Advisor extracts *per-transaction* demand traces: these provide information about resource usage on each transaction's *critical path*, essential to predicting response time. Given the thread events and request markers, it ascribes each event and each cycle of computation to exactly one transaction or background task. It then groups transactions according to stored procedure invocation: for example, a TPC-C "new order" transaction invokes the stored procedure `tpcc_neworder` exactly once.

3.3. Virtual performance counters

In addition to its use in answering "what-if" questions, detailed end-to-end tracing provides a wealth of information about current system performance, which can be represented in a variety of ways to help DBAs understand system behavior. Once we have a workload trace, we can easily construct new performance views as needed. Figure 3 shows one such view: a stack depth analysis [18] of buffer cache locality by transaction type. In addition to summary views, it is often instructive to examine in detail the behavior of a single transaction. For example, "timeline views" such as Figure 2 can be automatically generated from the per-transaction traces.

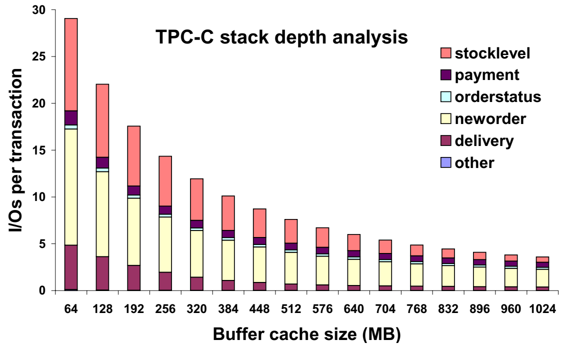


Figure 3. Stack depth visualization

4. Answering "what-if" questions

The Resource Advisor's primary function is, based on a workload demand trace, to answer "what-if" questions about system performance in a hypothetical hardware configuration. This section describes the performance metrics of interest — throughput and response time — and the operational formulas used to predict them. The inputs to these formulas are provided by the resource models, which predict the behavior of the CPU, storage, and buffer pool managers when resources change. The storage and buffer pool models are described in detail as they are affected by changes in buffer pool size, the resource that we vary in the experimental validation.

4.1. Performance metrics

The *throughput* of a DBMS running some workload is the number of transaction requests satisfied per second. It depends not only on server performance but also on the client request rate. If the client workload is closed-loop, each concurrent user has at most one outstanding transaction at any time, and waits for some *think time* between transactions. In an open-loop workload, request rate is independent of response time. *Saturation throughput* is the maximum achievable server throughput, when either the CPU or the I/O system is fully utilized.

Thus, the interesting metrics to predict are:

1. closed-loop throughput with a given think time
2. saturation throughput
3. response time

This work is concerned with server performance. Thus, "response time" consists only of the time between the server receiving a transaction request and sending out a response after committing or aborting the transaction. All delays external to the server process — client-side processing, network queuing, etc. — are considered to be part of "think time" from the server's point of view.

4.2. Throughput prediction

To predict the throughput of a workload, the Resource Advisor first identifies the bottleneck or throughput-limiting resource. The expected server throughput T will be that of the bottleneck resource, the lowest of:

- $T_{max/io}$: storage subsystem's service rate for the I/O request stream from the buffer pool manager.
- $T_{max/CPU}$: processor's execution rate for the workload's computational demands.
- $T_{max/workload}$: client request rate

$T_{max/io}$ is computed as as $\frac{1}{n_{io}t_{io}}$. Here n_{io} is the average number of I/O requests per transaction predicted by the buffer pool model (Section 4.4). t_{io} is the average I/O service time predicted by the storage model (Section 4.5).

$T_{max/CPU}$ is computed as $\frac{1}{t_{CPU}}$, where t_{CPU} is the average amount of CPU time used per transaction. The Resource Advisor currently assumes that this is inversely proportional to processor speed and independent of other hardware parameters.

$T_{max/workload}$ is the request rate T for an open loop, or $\frac{N_{users}}{t_{think}}$ for a closed loop where N_{users} is the number of concurrent users and t_{think} is the mean think time. The DBA must specify whether the expected workload is an open or a closed loop, and also the associated rate parameters (T, N_{users}, and t_{think}); these can be different from those of the currently observed workload, allowing "what-if" queries about changes in workload request rate and concurrency as well as resource changes.

4.3. Response-time prediction

Unlike throughput, response time depends not only on overall resource usage but also on the length of the critical path (Section 3.2). In general, predicting response time when resources change would require us to model the scheduling interactions between concurrent transactions. We avoid this hard problem by using an analytic approximation for OLTP workloads, which gives promising results despite its simplicity.

While throughput is by definition an aggregate workload measure, response time can also be measured for each transaction individually or aggregated by transaction category, where different categories may have very different response-time characteristics. The Resource Advisor currently categorizes transactions according to the stored procedures invoked and predicts the mean response time for each category. Depending on the DBMS and workload, many factors can contribute to response time, notably locking [6, 19]. Currently we only predict delays due to waiting/executing on hardware resources (CPU and I/O).

For each transaction type X, the Resource Advisor computes the critical-path CPU time per transaction and scales it using the CPU model. I/O blocking time is proportional to the number of blocking I/Os per transaction (predicted by the buffer pool model) and to the average delay per I/O including queuing delay. Delay per I/O is assumed to be inversely proportional to storage system idleness, and storage system utilization proportional to the total amount of I/O traffic. Thus the predicted mean response time for transaction type X is

$$t_X = t'_{X/cpu} + b_{X/io} d'_{io} \frac{1 - U'_{io}}{1 - U'_{io} \frac{n_{io}}{n'_{io}}}$$

where the primed variables are measured from the live system trace and the unprimed variables are predictions for the hypothetical case. $t'_{X/cpu}$, d'_{io}, U'_{io}, and n'_{io} are the mean critical-path computation time for type X, the mean delay per blocking I/O, the I/O subsystem utilization, and the average number of I/Os per transaction across all types. The buffer pool model provides $b_{X/io}$ and n_{io}: the predicted number of blocking I/Os per transaction of type X and the predicted number of I/Os per transaction across all types.

4.4. Buffer pool model

In the above analyses, the throughput and response time predictions depend on the amount of I/O generated per transaction. I/O is generated due to buffer cache misses and dirty page evictions, which depend on the buffer pool size and also on the buffer management strategies used by the DBMS, especially the cache eviction policy [11]. The prototype Resource Advisor uses a cache simulator specific to the DBMS under study, with a globally shared buffer cache and an LFU eviction policy. Our model assumes that the sequence of buffer references, as well as memory allocations for purposes other than caching disk pages (temporary objects or working memory), is independent of the underlying buffer pool size. We have confirmed these assumptions through code inspection and experimentation.

The Resource Advisor makes three simplifications in modeling the real buffer manager. First, it only models the I/Os to the main database tables, assuming the most common DB configuration where the recovery log is on a separate disk and is not the performance-limiting factor. Second, it ignores the opportunistic writeback of dirty pages by the DBMS, which occurs only when the I/O subsystem is idle and does not affect the analysis of I/O as a performance-limiting factor. Finally, the DBMS replenishes a small free buffer pool in the background whereas the simulator evicts pages strictly on demand, a negligible difference given that during steady-state operation the free pool is very small relative to the total memory.

4.5. Storage model

The storage model predicts the performance of the I/O request stream predicted by the buffer pool model. It uses an analytic model [24] based on the Shortest Seek Time First (SSTF) scheduling algorithm used by almost all modern disk device drivers. It assumes that the I/O request stream is random, with little or no spatial locality, a good fit for highly concurrent workloads such as OLTP with many independent requests, especially as the buffer cache absorbs much of the locality in the access pattern. If data pages are distributed across disks, the Resource Advisor separates out the I/O requests by disk and analyzes each disk individually as a potential bottleneck. It does not currently model more complex storage schemes such as mirroring or RAID, but such models can easily be plugged in to the framework.

The SSTF model predicts the mean I/O service time t_{io} as a function of the known disk parameters such as number of cylinders, seek times, etc. and also of the mean I/O queue length q_{ave}. The queue length is an input to the model and must be predicted, since it can depend on the resource availability. For example, larger buffer pools generate fewer buffer cache misses, which leads to fewer outstanding I/Os per user and hence shorter queues. We use

$$q_{ave} = (N_{users} - N_{cpu} - t_{think}T)(\frac{N_{nonblocking}}{N_{blocking}} + 1)$$

The first term represents the average number of user connections blocked in I/O at any given time, and the second represents the average number of outstanding I/Os for each such connection. N_{users} and t_{think} are the number of user connections and the mean think time, both workload parameters specified by the DBA. N_{cpu} is the expected number of running or runnable transaction threads, which for our non-preemptive CPU scheduler is the number of processors in the system. The ratio $\frac{N_{nonblocking}}{N_{blocking}}$ of non-blocking I/Os (readaheads and writebacks) to blocking I/Os (demand reads) is predicted by the buffer pool model.

These equations give us the I/O service time t_{io} as a function of the transaction rate T. For a closed-loop, I/O-bound workload, the transaction rate in turn is a function of t_{io} (Section 4.2). The Resource Advisor solves the mutual equations numerically using an iterative computation. Section 5 shows that this simple model of queue length, combined with the analytic storage model, accurately tracks the changes in I/O subsystem performance with changes in buffer pool size.

5. Evaluation

In evaluating the Resource Advisor, the high-level question to be answered is:

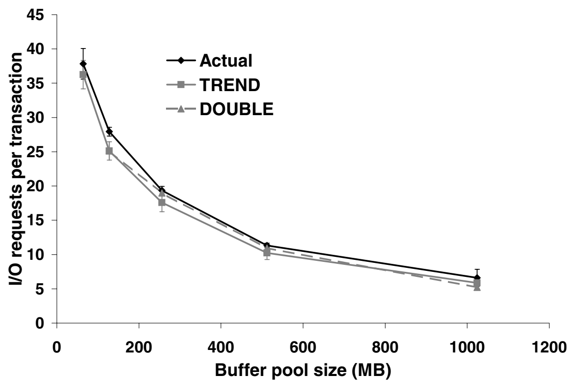


Figure 4. I/Os per transaction for SAT

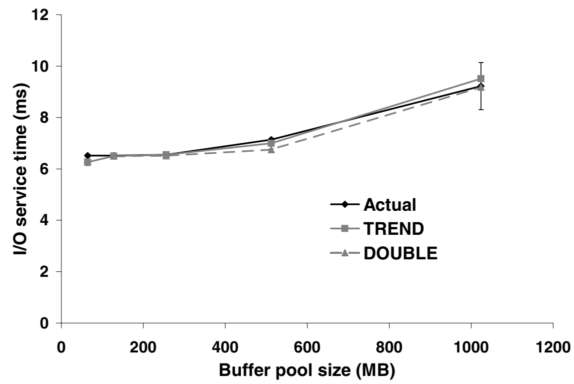


Figure 5. I/O service time for SAT

Given a live system running a workload, can the Resource Advisor efficiently and accurately predict the performance of the same workload with different resources?

In this evaluation, we answer this question for an OLTP workload with the varying resource being buffer pool size, i.e. memory availability. We break down our earlier question into four sub-questions, and answer each in turn:

- For a closed-loop workload, can the Resource Advisor predict throughput at different buffer pool sizes?
- Given a non-saturation workload, can it predict saturation throughput at different buffer pool sizes? More generally, can it predict the effect of changes in workload request rate, or from an open to a closed loop?
- Given a non-saturation workload, can it predict response time at different buffer pool sizes?
- What are the runtime overheads and other costs of deploying the Resource Advisor?

5.1. Workloads and experimental setup

Our evaluation uses two variants of a TPC-C [25] workload, which differ only in the transaction request rate:

- SAT is a closed-loop saturation workload. Each user operates in a closed loop with near-zero think time, placing the server under heavy load.
- OPEN is an open-loop non-saturation workload with a low, constant transaction rate such that the server always has significant amounts of idle time.

All the experiments used a single instance of SQL Server running on Windows Server 2003 on a single 2.7 GHz Intel Xeon processor. The 10GB database was stored on a single 80 GB Western Digital WDC800JB disk, with the transaction log and event trace log on two additional disks. Our aim was not to obtain optimal performance from the DBMS server, but to predict the change in performance when resources change. Thus, we opted for simplicity rather than careful tuning of the hardware and DBMS configuration.

Our stress client simulates 200 independent user connections to the server, runs on a single 2.8 GHz Intel Pentium processor, and is connected to the server by a 100 Mbps ethernet switch. Each workload run consists of at least 5000 transactions and is preceded by a warm-up phase of at least 30000 transactions. Each run is repeated with buffer pool sizes of 64 MB, 128 MB, 256 MB, 512 MB, and 1024 MB. The CPU and disks were unchanged in all cases.

The aim of each experiment is to validate the Resource Advisor's answer to a "what-if" question against the measured result of carrying out the hypothetical change. Of the many "what-if" questions a DB administrator might ask, we chose two typical ones related to buffer pool memory:

- What will the performance be if I double the memory?
- What is the trend as I continue to add memory?

These questions are answered by evaluating two predictors:

- DOUBLE predicts performance when the buffer pool size is doubled. It predicts the performance at 128 MB from the trace at 64 MB, at 256 MB from 128 MB, etc.
- TREND predicts performance over the entire range of buffer pool sizes, based on traces from the lowest-memory configuration (64 MB).

5.2. Closed-loop throughput prediction

This experiment evaluates the accuracy of DOUBLE and TREND in predicting the throughput of SAT. We evaluate the accuracy of each individual prediction component as well as the final result. The results are based on 5 identical runs at each configuration, showing the mean and standard deviation of both the measured and the predicted values.

Our first step is to evaluate the accuracy of the buffer pool model. The predicted value is the number of I/Os generated per transaction, which determines the I/O-bound throughput $T_{max/io}$. Figure 4 shows the predicted and actual I/Os per transaction. Figure 5 then shows how well the storage model predicts the I/O service time. Note that our

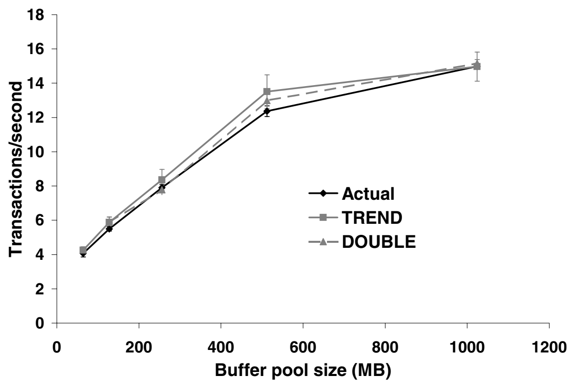


Figure 6. Throughput of SAT

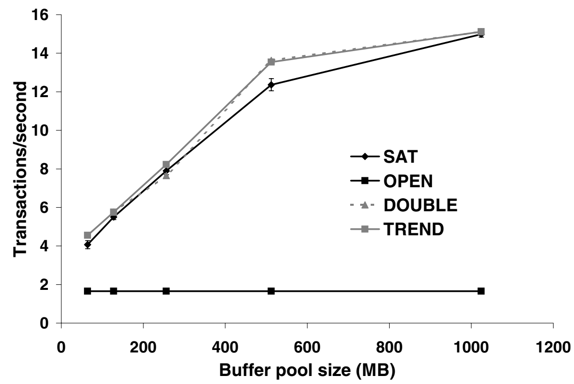


Figure 7. Throughput of SAT (from OPEN)

model correctly tracks the slight increase in service time due to decreasing queue lengths.

Given the I/O request stream and service time, the Resource Advisor predicts throughput using the analysis of Section 4.2. Figure 6 shows the accuracy of this prediction. Note that throughput was I/O-bound at all buffer pool sizes except at 1024 MB, when it becomes client-bound, i.e. limited by workload think time. In all cases, the predictor correctly identifies the bottleneck, and we see that TREND tracks the performance curve, while DOUBLE predicts throughput to within 7% or better.

5.3. Saturation throughput prediction

The previous experiment showed that that the Resource Advisor can predict throughput across changes in buffer pool size. Often we also want to predict the effect of changes in *request rate*. In this experiment, we predict the throughput of the high-rate, closed-loop SAT workload (characterized by its known think time) based on a trace of the low-rate open-loop OPEN workload. This tests the following capabilities of the Resource Advisor:

- *Predicting saturation throughput*: as SAT is a saturation workload in all but the 1024 MB case, correctly predicting its throughput corresponds to correctly predicting saturation throughput. At 1024 MB, it corresponds to correctly predicting that the bottleneck will move outside the server.
- *Predicting across arrival patterns*: SAT and OPEN differ in their request rates as well as their rate models (closed vs. open). Predicting the performance of one by observing the other will show that our models can predict across changes in request rate as well as arrival pattern.
- *Demand extraction*: the throughput prediction is based on the assumption that the demand trace captures all relevant features of the workload and is independent of server load. This is validated by using the demand trace of OPEN to predict performance under the very different load regime of SAT.

Figure 7 shows the measured throughput of SAT, the DOUBLE and TREND predictions for SAT based on a single run of OPEN, and the throughput of OPEN itself. Although SAT and OPEN behave very differently, the Resource Advisor correctly infers the former by observing the latter: DOUBLE predicts the saturated throughput to within 10% or better and correctly identifies the client as the bottleneck for the 1024 MB case.

5.4. Open-loop response time prediction

Recall from Section 4.3 that the Resource Advisor groups transactions by type. This lets us measure and predict mean response time by transaction type, since different transaction types have very different critical paths. We predicted the mean response time for each of the five TPC-C transaction types, with prediction errors of DOUBLE varying from 33%–68%. Although this appears large, it is important to note that the underlying variation in response time is also large: in 19 of 20 cases, the error was smaller than the observed standard deviation. This indicates that there is little room for further improvement in predicting mean response time. Instead, the focus should be on predicting the frequency distribution of response times.

Prediction error is also small compared to the change in response time as memory size is increased. Across the full range of memory sizes measured, response time of all five transaction type changes by more than an order of magnitude. Our predictions accurately follow this *trend* for all five transaction types. Figure 8 shows this result graphically for the "new order" type.

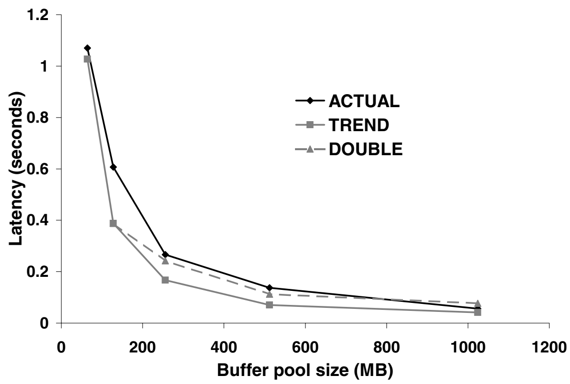


Figure 8. Response time in OPEN

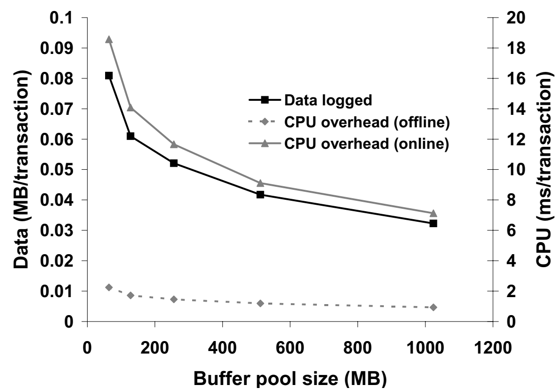


Figure 9. Overhead of event logging

5.5. Resource Advisor overheads

Figure 9 shows the runtime overheads of the Resource Advisor: the amount of trace data generated and the CPU consumed per transaction. The overhead per transaction is higher at lower buffer pool sizes: there are more I/O events to process, as well as more background events since transaction execution times are lengthened.

The worst-case CPU overhead is 6.2%, for an unoptimized C# implementation of the Resource Advisor running online. With offline operation, this is reduced to 1.2% for tracing and logging. The trace data rate is a modest 0.44 MB/s in the worst case, whether for online consumption or logging to disk. These overheads could be reduced further with a more optimized implementation, and by using a sampling approach rather than always-on tracing.

A potential concern with self-prediction is the amount of effort required to add it to an existing DBMS. We advocate that future DBMS designs incorporate self-prediction as a goal from the beginning; however, our experience is that this capability can also be added to legacy code with a modest amount of work. Instrumentation of the DBMS required only 189 additional lines of code in 6 source files. The rest of the Resource Advisor runs as a stand-alone program, and is also modest in size: 1150 lines of code.

6. Related work

Much research has identified buffer memory as a key resource for database throughput and response time [9, 11] and proposed various techniques to optimize or adapt memory usage: for example, by dynamically limiting the working memory allocated per query [5]. The Resource Advisor complements this work by predicting the behavior of the buffer manager at different memory sizes, given the memory allocation and buffer references made by higher layers.

Our buffer model is based on cache simulation, an old and well-studied idea [10, 18]. An alternative approach is to model the cache hit ratio of a specific workload such as TPC-C as a function of buffer size [15, 26]. Although we use a TPC-C like workload in our evaluation, the Resource Advisor itself is workload-agnostic, using live system traces to capture workload characteristics, and usable by DBAs with little or no understanding of the workload.

Several studies have proposed detailed cost models of CPU usage and of the processor cache hierarchy [2, 4, 17]. Others have developed models of storage device performance [24, 27]. Our contribution is in integrating these approaches into a broader mechanism to answer "what-if" questions about hypothetical hardware changes.

Magpie [3] and Pinpoint [8] both use end-to-end event tracing, for workload modeling and fault detection respectively. The Resource Advisor uses the same principle for DBMS performance prediction and capacity planning.

Finally, resource provisioning is only one of many database configuration and maintenance tasks. Orthogonally to our work, recent research has focused on automated physical database design tools to reduce manual intervention and maximize performance. The Database Tuning Advisor [1] and DB2 Advisor [16] suggest the most appropriate set of indexes and materialized views as well as the best physical layout of tables. AutoPart [23] automates schema design using data partitioning on large-scale datasets.

7. Conclusion

This paper presented a design and implementation for a database *Resource Advisor* that predicts the performance impact of changing resource availability. It is based on fine-grained, low-overhead tracing; per-transaction demand extraction; and simple, lightweight, workload-agnostic resource and performance models.

Our primary contribution is to demonstrate that the Resource Advisor can accurately answer "what-if" questions about hypothetical resource changes for a live system: the key requirement for automating decisions on resource upgrades or reprovisioning. We validated this claim by predicting the throughput and response time of an OLTP workload as a function of buffer pool size. Our second contribution is a modular architecture with easily replaceable analysis and modeling components.Finally, we demonstrated the value of end-to-end transaction tracing both in response time prediction and in performance visualization.

Our current goal is a more thorough validation of the Resource Advisor prototype, for instance using DSS (Decision Support System) workloads which behave very differently from OLTP. We also intend to extend our models to predict response time distributions in addition to mean values, and also to predict the effect of changing the workload transaction mix. Finally, we are extending both the instrumentation and the models to other performance-limiting factors such as locking and network communication.

References

[1] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database tuning advisor for Microsoft SQL Server 2005. In *Proc. 30th VLDB conference*, Aug. 2004.

[2] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: Where does time go? In *Proc. 25th VLDB conference*, Sept. 1999.

[3] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier. Using Magpie for request extraction and workload modelling. In *Proc. 6th Symposium on Operating Systems Design and Implementation*, Dec. 2004.

[4] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *Proc. 25th Annual International Symposium on Computer Architecture*, June 1998.

[5] K. P. Brown, M. Mehta, M. J. Carey, and M. Livny. Towards automated performance tuning for complex workloads. In *Proc. 20th VLDB conference*, Sept. 1994.

[6] M. J. Carey, S. Krishnamurthy, and M. Livny. Load control for locking: the 'half-and-half' approach. In *ACM Symposium on Principles of Database Systems*, Apr. 1990.

[7] S. Chaudhuri and G. Weikum. Rethinking database system architecture: towards a self-tuning RISC-style database system. In *Proc. 26th VLDB conference*, Sept. 2000.

[8] M. Chen, E. Kiciman, E. Fratkin, E. Brewer, and A. Fox. Pinpoint: Problem determination in large, dynamic, Internet services. In *Proc. International Conference on Dependable Systems and Networks (IPDS Track)*, June 2002.

[9] H. Chou and D. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proc. 11th VLDB conference*, Aug. 1985.

[10] A. Dan, P. S. Yu, and J.-Y. Chung. Characterization of database access pattern for analytic prediction of buffer hit probability. *VLDB J.*, 4(1):127–154, 1995.

[11] W. Effelsberg and T. Härder. Principles of database buffer management. *ACM Transactions on Database Systems*, 9(4):560–595, Dec. 1984.

[12] J. Gray. *Benchmark handbook: for database and transaction processing systems*. Morgan Kaufmann, 1992.

[13] IBM. Autonomic computing: IBM's perspective on the state of information technology. `http://www.research.ibm.com/autonomic/manifesto/`, 2001.

[14] IBM. DB2 performance expert. `http://www-306.ibm.com/software/data/db2imstools/db2tools/db2pe/`, 2004.

[15] S. T. Leutenegger and D. Dias. A modeling study of the TPC-C benchmark. In *Proc. 1993 ACM SIGMOD International Conference on Management of Data*, May 1993.

[16] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, and A. Skelley. DB2 Advisor: An optimizer smart enough to recommend its own indexes. In *Proc. IEEE International Conference on Data Engineering (ICDE)*, Feb. 2000.

[17] S. Manegold, P. Boncz, and M. L. Kersten. Generic database cost models for hierarchical memory systems. In *Proc. 28th VLDB conference*, Aug. 2002.

[18] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970.

[19] D. T. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Proc. IEEE International Conference on Data Engineering (ICDE)*, Mar. 2004.

[20] Microsoft. Event Tracing for Windows (ETW). `http://msdn.microsoft.com/library/en-us/perfmon/base/event_tracing.asp`, 2002.

[21] Microsoft. Improving SQL Server performance. `http://msdn.microsoft.com/library/`, 2002.

[22] Oracle. Oracle database manageability. `http://www.oracle.com/technology/products/manageability/`, 2004.

[23] S. Papadomanolakis and A. Ailamaki. Automating schema design for large scientific databases using data partitioning. In *Proc. 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, June 2004.

[24] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Proc. 1990 Usenix Winter conference*, Jan. 1990.

[25] Transaction Processing Performance Council. *TPC Benchmark C (On-Line Transaction Processing) Specification*. `http://www.tpc.org/tpcc/`.

[26] T.-F. Tsuei, A. N. Packer, and K.-T. Ko. Database buffer size investigation for OLTP workloads. In *Proc. 1997 ACM SIGMOD international conference on Management of Data*, May 1997.

[27] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with CART models. In *Proc. 12th Annual Meeting of the IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Oct. 2004.