Protecting the Quality of Service of Existing Information Systems *

Kevin S. Beyer Miron Livny Raghu Ramakrishnan
Computer Sciences Department
University of Wisconsin, Madison WI 53706, USA
beyer,miron,raghu@cs.wisc.edu

Abstract

Organizations that offer external access to their data need a mechanism that ensures a desired level of service for local users. We propose such a mechanism, called the provider agent (PA) architecture, that protects local users by ensuring a (DBA specified) quality of service for local requests in the face of computational demands made by external requests. The PA is a general purpose solution that enhances most information systems currently available. The novelty of our approach is the combination of request profiling with load control mechanisms to improve both protection and performance, while not requiring any modifications to the underlying information system. We demonstrate the effectiveness of the proposed techniques with a prototype PA for a commercial DBMS.

1. Introduction

People want access to information across organizational boundaries, and information systems are becoming increasingly accessible through integration technology. However, each system is usually owned by one organization, and has a primary purpose that is more important than this new, external access. For example, accessing a customer's order information during a sales call is probably more pressing than determining the number of widgets sold last month.

In this environment, the local users are recognized as the owners of the system, and the external requests should be processed only when the owners allow it. If the site does not take precautions, then the external workload could easily disrupt the local service. The damage caused by a heavy external workload is usually a large increase in local response time, but the damage could also be unforeseen failures of local requests due to limited resources (e.g., a lack of memory or connections).

One way to keep external activity from interfering with local users is to replicate the existing system. The external requests cannot interfere with local users since they are not using the same system. However, this solution is not always acceptable for the following reasons. First, the cost of creating and administering a second system could be prohibitive. Second, the external users might need access to up-to-date information. Third, a demand is still placed on the local users system when the replicated system is periodically updated, and that demand, although more predictable, still needs to be scheduled. Fourth, if the external requests are prioritized, replicating the system for each priority level is infeasible. Finally, replicating the system partitions the computing resources, so one system could be idle while the other is overloaded. For these reasons, we focus on an unreplicated solution.

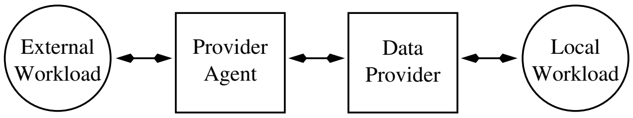

Figure 1. Provider Agent Architecture

This paper describes the architecture of a system, called the Provider Agent (PA), that protects the local quality of service of a data provider in the face of external processing demands. The PA mediates between the provider and the external world (see Figure 1), and extends the traditional integration "wrapper" (e.g., [4, 23]) with resource management. The architecture is designed to work with any type of data provider available today, for example, a DBMS, a Web server, an FTP server, or even data generating programs like a simulation system. The PA does not mandate any changes to the underlying data provider or the existing (local) applications.

In designing the PA, we used request profiling and dynamic load control techniques to protect the local users. Load control transforms the worst case external workload from disastrous to predictable and acceptable. Profiling enables the administrator to specify load controls based upon

*This work was supported in part by ORD contract 144-ET33, a Presidential Young Investigator award, and NASA research grant NAGW-3921.

query resource requirements rather than treating all requests the same, regardless of the impact of the request.

To evaluate the effectiveness of these techniques, we implemented a prototype PA for a relational DBMS. We found that our load control mechanisms, which are implemented outside the DBMS, effectively maintain the local quality of service at a predefined level, and our profiling technique dramatically improves the throughput of mixed external workloads.

In the remainder of this paper we further motivate the PA by describing some potential applications in Section 2. Section 3 discusses our measure of success. In Section 4 we detail the PA load control and profiling mechanisms. Section 5 describes an implementation of the PA for a commercial DBMS and Section 6 presents the performance results. The remaining sections describe related work (Section 7), future work (Section 8), and conclusions (Section 9).

2. Applications

The PA is applicable to a broad range of scenarios. For example, when an enterprise connects its disparate departmental databases into a multi-database, the PA would ensure that the large ad-hoc queries posed to the multi-database by someone in the marketing department do not overwhelm any of the underlying production databases. When a company offers order tracking information to Internet users, the PA prevents the status requests from disrupting order processing.

For a non-database example, consider a web server that is about to be searched by an automatic indexing agent. If the agent arrives during a peak period of the day, it would be useful to delay its requests until an off-peak time or at least control the rate at which the it accesses the data. The service provided by the indexing sites is invaluable to web users that are trying to locate information, so simply rejecting the request is probably not acceptable. The current solution is to rely upon the agent writers to space requests to an individual server [15, 8]. The first problem with this approach is that the web site administrator must rely on the agent writer to be conscientious and careful when writing it. The second problem is that at different times during the day, the server might be able to process the requests at a much higher rate. Why should the agent make arbitrary decisions about when and how frequently the web server is accessed when the server is much more capable to answer these questions? If the indexing agent were to access the web server through a PA, then the PA would ensure that the agent does not drastically interfere with the web servers response to normal requests.

3. Quality of Service

The primary concern in this paper is the quality of service (QoS) offered to the local users. Although many QoS measures exist, we use changes in the response times experienced by the local users to illustrate the protection offered by a particular PA policy. Instead of summarizing the response times into a single number like the average response time, we use the cumulative distribution of the local response times as our QoS measure.

Only when the local users are satisfied can the PA consider improvements in external processing. Two scheduling policies cannot be compared on the basis of external performance without considering the local QoS. In comparing external performance, we use the throughput rather than response time because we are more interested in the amount of work completed than how quickly the work was performed.

4. Mechanisms

The PA achieves its objectives through a variety of load control mechanisms. Each of the mechanisms presented in this section describes a way that the PA can control when and how a particular external job is run. The merits of load control are well known, dating back to operating systems work of the '60s and '70s [11, 6]. The novelty of our approach is in extending these ideas to the protection of the local QoS in the presence of an unknown external workload without changing the existing system, in our definition of fractional MPL, and in combining load control with query profiling.

4.1. Integer MPL

Adjusting the multiprogramming level (MPL), by definition, controls the maximum number of active requests in a system. By limiting the number of external jobs, the DBA controls the worst case performance of the local workload. Theoretically, by constraining the external workload to a finite MPL, a stable local workload will never become unstable by running external jobs (i.e., the number of outstanding local requests will not go to infinity). Unfortunately, the number of local requests in the system, will continue to increase until the system again stabilizes, so the local response times can become unacceptable or the system can run out of some critical resource (e.g., network connections).

MPL is a simple and effective control because it smoothes the external service demands by queuing requests outside of the provider, but it has several weaknesses. First, MPL is an integer value, so the impact on the local users cannot be fine-tuned, and in particular, if an MPL of one causes too great of an impact, then external queries cannot be processed (see Sections 4.2 to 4.4). Second, MPL does not consider which resources will be affected, so the DBA must plan for the worst case of all the external requests accessing the same device (see Section 4.5). Third, small external requests get poor response times because they are forced to wait behind large requests (see Section 4.5). Finally, slow external consumers tie up provider connections and delay the processing of other external requests (see Section 4.6).

4.2. Fractional MPL via Spacing

If the response times of the local requests are still unacceptable even after an MPL is applied, the external requests can be further subdued by spacing job executions. Spacing jobs reduces the *average* MPL, and thus allows a fractional MPL to be specified. Spacing jobs further reduces the amount of time that local jobs must wait due to external jobs and gives the system time to recover from any short-term resource deficit.

The interval between jobs can be specified in several ways. In its simplest form, a space can be a fixed time period. For example, after every job, pause for 10 seconds. A second way to define the space is based upon the amount of resources consumed by the job, which means that the pause after large jobs is longer than after short jobs. One last way to define spacing is based upon the amount of time the job spends in the provider. For example, if the space is specified as 100% and a job executes for 30 seconds, then the PA would wait an additional 30 seconds before allowing another job to enter the system. This last type of spacing not only responds to the length of the job, but it also responds to how busy the system is. Another advantage of this definition of spacing is that it correlates directly with MPL; the effective MPL is defined by:

$$MPL_{eff} = \frac{MPL_{act}}{spacing + 1}$$

For example, an MPL of 0.5 can be achieved with an MPL of 1 and a 100% spacing, and an MPL of 1.5 can be achieved with an MPL of 2 and a 33% spacing.

However, the effect of spacing is not identical to that of MPL, because the pressure on the system oscillates. While an external job is running, the local jobs experience higher response times, and while no job is running, the response times are lower. This oscillation implies that an MPL of two with 100% spacing does not control the system in the same way as an MPL of one (but in either case, the average number of external requests running is one).

4.3. Suspending Jobs

Instead of spacing between external jobs, the jobs can be run more slowly by placing the spaces between sub-job units of work. The advantage of this technique is that a large job behaves like a group of small jobs, therefore the average local response time is achieved in a smaller time period. This technique also allows external jobs to use a processor sharing scheduling policy, although the quanta would be relatively large. We identified three ways of slowing down a job: intra-job spacing; suspending jobs; and dividing jobs. The effectiveness of these techniques depends upon the type of provider and the type of requests.

Intra-job spacing places an idle period between every block of the result. For example, the space could be placed between every 100 tuples fetched from a DBMS. If a result buffer exists between the provider and the PA, then intra-job spacing might not slow the external requests at all because the provider is filling the buffer while the job is sleeping, but the effect can be diminished by using larger block sizes. A more difficult problem with using intra-job spacing is the amount of data in the result might not correspond with the amount of work needed to find the result. For example, if an aggregate is applied to a large query, the result is one tuple but the query might execute for an hour. The spacing should be based upon the amount of work done at the provider, rather than the size of the result.

Intra-job spacing can be generalized to job suspensions. When the system load is high, suspend the job until the system recovers. If a job is monopolizing an MPL unit for too long, suspend it and let another job run instead. Unfortunately, many providers offer no way to suspend a job except when its result buffer is filled. A provider with a feature like the DB2 Governor Facility could assist the PA by automatically suspending the job after each unit of work.

Another technique that the PA can use to slow the execution of a job is to break the job into smaller jobs. The smaller jobs can then be scheduled using job spacing. Unfortunately, dividing jobs is often difficult or impossible.

The major drawback to all of these techniques is that they hold resources, like disk and memory buffers, locks, and connections, for a longer period of time. If a suspended job were holding read locks on some data and a local job wanted to update that data, then suspending the job actually slows down the local job. This problem still exists even when dividing the job into smaller jobs. If each of the smaller jobs are executed in a separate transaction, the combined answer could be inconsistent, so the jobs should be run in one transaction which implies that locks will be held between job executions.

4.4. Feedback

In this section, we describe a feedback mechanism that monitors the utilization of key resources, for example, the disk drives, CPU, network, and memory, to decided when to start and stop jobs. This mechanism can detect periods of relatively low activity, similar to the way the Condor system [16] finds idle resources to run batch programs. If the local workload experiences periods of high and low activity throughout the day, the PA can run external jobs during the periods of low activity. If an external job is running when the system is experiencing a heavy load, the PA can kill or suspend the job and restart it when the load decreases.

The feedback data is used to define the spacing of external jobs. External jobs are allowed to start only when the utilization of the busiest device is below some threshold. The mechanism can be extended so that if the utilization of some device exceeds another threshold, an external job is suspended or killed. The advantage of the feedback mechanism is that it responds to changes in the local workload. For example, if the system is under a heavy load, feedback will keep the PA from starting a job, while spacing must occasionally start a job to determine that the system is indeed still busy.

The feedback mechanism is controlled by three parameters: the threshold that determines when to take action, the sampling interval, and the moving average time window. The utilization is sampled from the provider or the operating system by the PA once per sampling interval and averaged with the last few samples.

The sampling interval controls the resolution of decisions within the PA. As the sampling interval gets smaller, the PA get a more detailed view of the system. If the sampling interval is set too low, the overhead of sampling will adversely affect performance. Conversely, if the sampling interval is too high, the PA becomes sluggish because as far as the PA is concerned, the state of the provider has not changed. The proper setting for the sampling interval depends upon the duration of the external jobs and the volatility of the local workload.

The moving average time window controls how responsive the PA is to changes in the provider. A small time window means the PA will react quickly, but if the window is too small the PA tends to over-react. If the time window is longer, the PA is more tentative and responds slowly to both increases and decreases in utilization. When using feedback to detect changes in the workload, the moving average time window should be long enough to get a reasonable estimate of the true average. If the goal is to detect when no local jobs are using the system, the window should be small, perhaps eliminating the moving average altogether. When controlling the job spacing, the size of the window depends upon the duration of the external jobs. The window should be long enough to at least cover the execution of the job and the desired spacing.

4.5. Profiling

Information about the potential resource consumption of an external job is invaluable to the PA, but when an external requests arrives it is not tagged with any such information. As discussed in Section 4.1, one of the major drawbacks of limiting the MPL is that small jobs are forced to wait behind large jobs. But by using the job profile information, the PA can schedule small jobs differently from large jobs. From the perspective of protecting the local users, scheduling small and large jobs differently is intuitively reasonable because a single small job is not as likely to seriously impact the performance of the system.

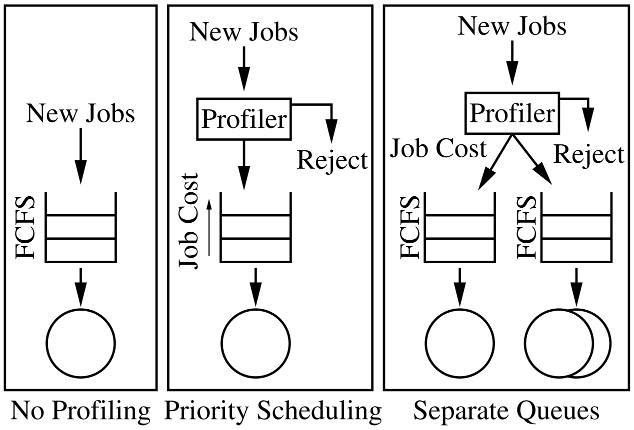

Figure 2. Queuing Strategies

The profile information can be used in a variety of ways (see Figure 2). One possible strategy groups jobs that perform less than 10 I/Os into one group, between 10 and 10,000 into a second group, and rejects any job that is expected to do more than 10,000 I/Os. Each group is given its own queue and scheduling policy (MPL, spacing, etc.). The maximum MPL in this case is the sum of the MPLs for each group.

Another strategy uses a single queue and scheduling policy, but orders the queue by the number of expected I/Os. With this strategy, a small job would wait behind at most one job that was longer than it (because the longer job was already executing when the small job arrived). If the MPL were more than one the small job must wait for only one job to complete.

Notice that with both of the strategies, absolute accuracy of the profile is not necessary. As long as the profiles are accurate relative to one another, both schemes function properly, except when a job is rejected. Occasionally, a profile could be completely wrong. The PA must ensure that the use of an incorrect profile will not detrimentally affect the protection that it provides. One way to provide insurance against running a job with a bad profile is to place a time limit on the execution of a job based upon the job's profile. The PA should also allow the administrator to accept a request that was mistakenly rejected.

When the profile contains details regarding the individual resources that a job is expected to use, the PA can more fully utilize the the providers resources. Either of the above strategies could be extended, for example, with a policy that allowed up to 5 jobs to run concurrently (MPL=5) as long no two jobs accessed the same disk drive.

Many data providers estimate the cost of executing a job without actually executing it. For example, a DBMS estimates plan costs during optimization. Unfortunately, this information is usually not exported by the provider — many DBMSs discard the cost estimates once the plan is produced. The PA could use the plan and reapply the statistics that the DBMS used to reconstruct the profile, but often the statistics are also not exported.

A work-around to these limitations is to replicate the profiler in the PA, similar to the approach taken in the Pegasus project at HP [7]. The profiler will frequently consult catalog information, so to avoid placing an additional strain on the provider, all of the needed catalog information should also be replicated within the PA. The advantage of using the provider's profile is that no effort is duplicated and the profile will generally be more up-to-date than the PA's profile, for example when an index is added or dropped but the PA has not been updated. When using the provider to produce the profile, the profiling should also be scheduled.

4.6. Buffering

Until now, we have not considered the rate at which the external users consume their results. We implicitly assumed that the results were consumed as quickly as they were produced. What happens when results are consumed slowly? Assume that the only control was an MPL of one. If one job decided not to consume any of its results for a while, the PA could not process any additional requests and the job would be holding resources (e.g., locks) for a long time.

One way to limit the impact of a slow consumer is to enforce a maximum execution time or a minimum transfer rate. If the constraints are violated, the job is suspended or killed and another job takes its place. A more forgiving strategy places a buffer between the PA and the external client. If the buffer is large enough, the PA can pull the entire result out of the provider as quickly as possible and allow the job to proceed as slowly as it wishes. The provider is no longer impacted by the slow consumer, and the PA can execute another job. Of course, the same problem could happen with the next job, and eventually the PA would run out of buffer space. When faced with this situation, the PA must either kill one of the slow jobs and reclaim its buffer space, or the PA must stop executing jobs and wait for enough buffer space to clear.

The buffer between the PA and the client can be both main memory and disk pages. All of the buffer space can be pooled together and shared among all of the PA clients. The page size should be relatively large since all I/O within one connection will be sequential. When swapping out a page to disk within one connection, the most recently filled page should be sent to disk. When choosing a page to swap out from all connections, the page least likely to be accessed should be swapped out. To find the least likely page, let r_i denote the rate at which connection i is consuming pages, and d_i denote the distance of the last page of connection i from the head of the queue. Then $t_i = d_i/r_i$ denotes the expected time until the last page is referenced. The least likely page to be accessed is the page with the maximum expected reference time.

4.7. External Priorities

The PA can be extended to support priorities among external requests. The techniques used to enforce priorities are similar to the techniques used in Section 4.5 to take advantage of profile information. One queue ordered by priority can be used to essentially achieve absolute priorities. Multiple queues, one per priority, with the total MPL divided among the queues based upon priority achieves relative priorities. The other mechanisms can also be extended with priorities. For example, when a job must be suspended or killed, choose the job with the lowest priority.

4.8. Provider Specific

Each provider comes with its own set of features that could be used to assist the PA in its task. For example, some providers have separate buffer pools based upon user-id, some have prefetching, and others have priority scheduling. We encourage the use of these features, but do not discuss them further.

5. Implementation

We implemented a prototype PA to experiment with the mechanisms and evaluate policies described in the previous section. This section describes the design decisions we made while implementing the PA for a commercial relational DBMS. Figure 3 depicts basic structure of the PA, and the components of the PA are described below.

Profiler: When a request arrives at the PA, it is profiled to estimate its resource needs and the expected impact that this

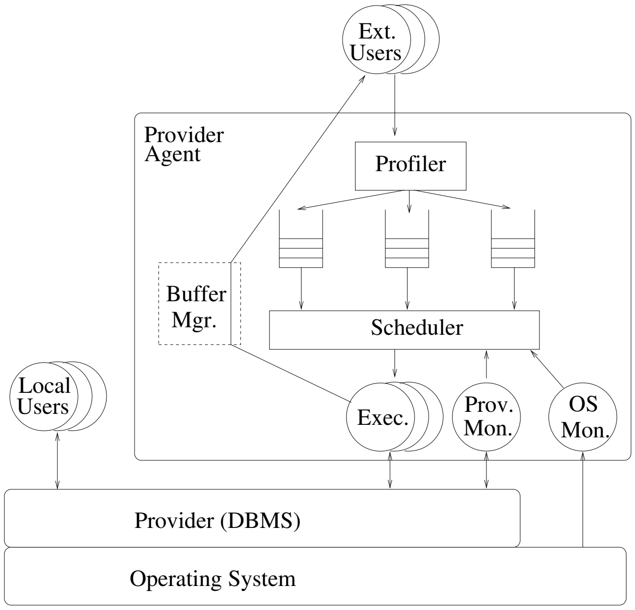

Figure 3. Provider Agent Architecture

request will have on the provider. The profiler is essentially a query optimizer designed to mimic the provider's optimizer. We chose to replicate the query optimizer inside the PA because the DBMS that we considered did not export the cost estimate. The goal of the profiler is not to find the best plan, but the plan that is expected to be produced by the provider. If the PA maintains additional statistics that the provider does not consider, the statistics should not be used during plan generation, but once the plan is chosen the PA can use the additional statistics to produce a better profile.

The profiler will frequently consult catalog information. So to avoid placing an additional strain on the provider, we cached all of the catalog information within the PA. Although not implemented in our prototype, the PA must have some mechanism for updating its statistics and becoming aware of new database objects. The mechanism could be automatic updates, or the administrator could update the catalogs as needed.

Scheduler: After a request is profiled, it is queued until the scheduler decides that the request can be safely executed without adversely affecting the local requests. This component uses the mechanisms described in Section 4 to enforce the policies established by the administrator, and in the next section, we evaluate the effectiveness of several possible schedulers.

Executor: An executor is the provider client that processes requests on behalf of the PA and its external clients. The executor contacts the external client to inform it that its job is ready to be processed. The executor then sends the request to the provider, receives the result, and sends it to the client, either directly or through the buffer manager. Our current prototype does not include a buffer manager, and in our experiments, external clients consumed results as quickly as possible.

System Monitors: The monitors periodically sample statistics provided by the operating system and the provider to inform the scheduler about current resource utilizations. The OS monitor was the only component of the PA that ran on the same machine as the provider because the operating system we used did not allow remote applications to directly obtain OS statistics.

6. Experiments

In this section, we evaluate the effectiveness of the key mechanisms of the prototype PA. We demonstrate the need for local QoS protection by running a moderately heavy external workload without the PA. Although we can disrupt local processing to an arbitrary degree, we show that even a moderate load can cause a 20 times increase in average response times.

Table 1. Experiment Parameters

Class	Parameter	Setting
local	#terminals	40
	inter-arrivals	exp(0.2) sec.
	queries	10 compiled point selects with non-clustered index
external	MPL	0 - 20
	#jobs	infinite
	queries	range count, 25-75% of relation with clustered index
system	#disks	1
	#relations	7
	tuple size	208 bytes
	relation size	12MB
	data cache	7.4MB of 1 page blocks 5.0MB of 8 page blocks
	duration	25 minutes

We then add the PA and show that load control with spacing allows the DBA to set the worst-case performance of the local workload. Once we establish that the basic load control mechanism is effective at controlling the performance degradation, we improve the PA by using a dynamic load control mechanism. We show that the dynamic control responds to the local workload by allowing more external jobs to be processed when the system is under-utilized. Lastly, we demonstrate that profiling dramatically improves the throughput of small external jobs in a mixed external workload.

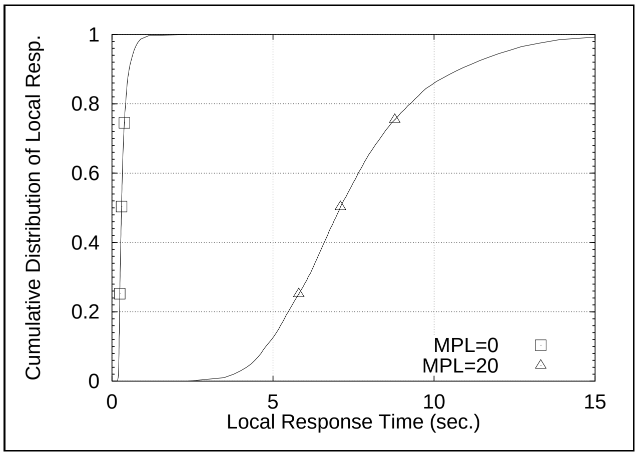

Figure 4. Impact of heavy external load

We generated a synthetic workload for the experiments, but we used our prototype PA and a current version of a commercial DBMS to ensure that our results were realistic. The local workload modeled a transaction processing system. We ran one experiment that performed updates to verify that updates did not unduly affect our results, but in the rest of the experiments the transactions were read-only. The external workload modeled exploration queries that needed to perform scans over large portions of the data. The experiment parameters are summarized in Table 1.

6.1. Experiment 1: No Control

To demonstrate the impact that external requests can have on the local response times, we ran a constant external workload of 20 queries. Figure 4 shows the cumulative distributions of the local response times over the 25 minute experiment. An MPL of 20 increased in average response over 20 times more than the response when no external jobs were run. When we pushed the system harder, we inadvertently caused a large percentage of local requests to fail because of a lack of memory needed to run the queries. Notice even a small period of high external activity is enough to wreak havoc on the hapless local users.

6.2. Experiment 2: Fixed MPL

In Figure 5, we show that the local response time can be controlled by adjusting the average number of external requests in the system via MPL and spacing. Decreasing the MPL shifts the entire distribution of response times towards the response time curve when no external queries were run (MPL=0). For this particular external workload, increasing the MPL beyond 1 actually decreases the throughput of the external requests because the competition between external

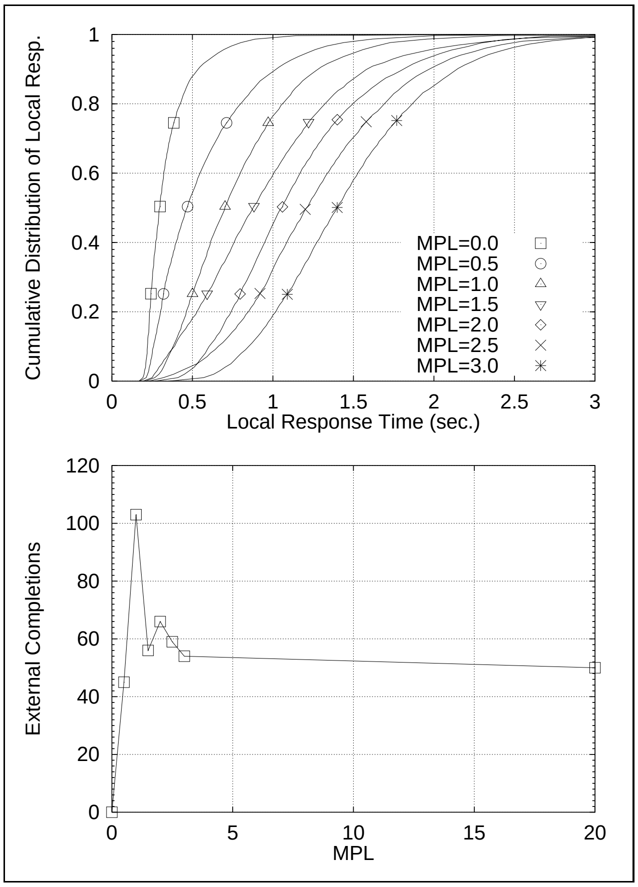

Figure 5. Fixed MPL Control

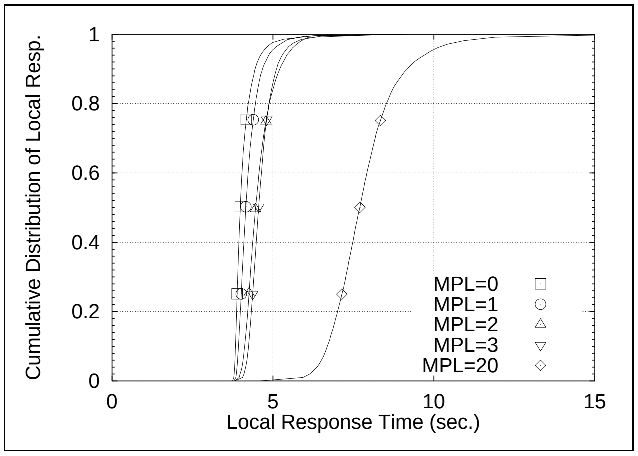

Figure 6. Fixed MPL with Updates

jobs changes the disk access pattern from sequential to random access.

We ran a similar experiment except that each local transaction updated the last record read after a 3 second delay. Figure 6 shows that the external MPL still controls the lo-

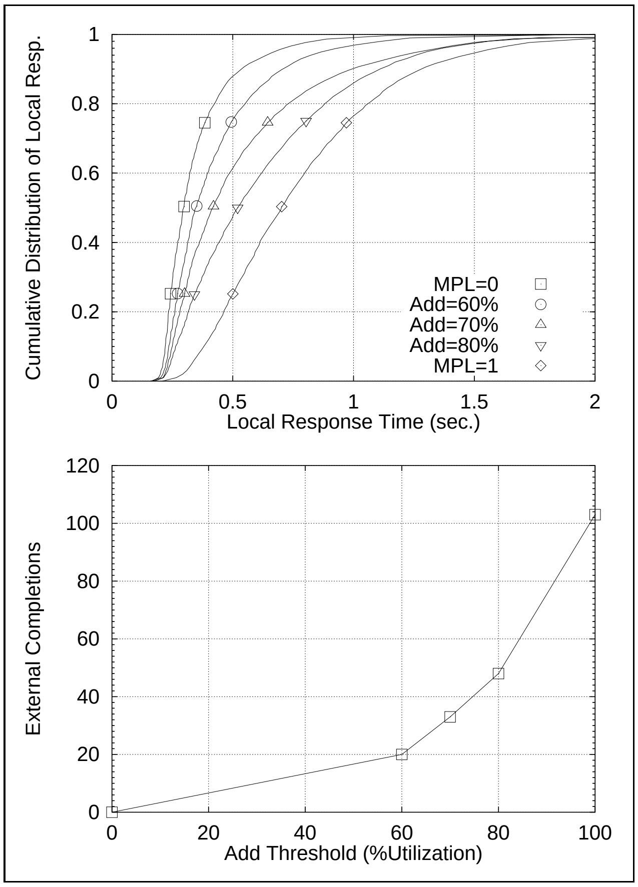

Figure 7. Dynamic MPL Control

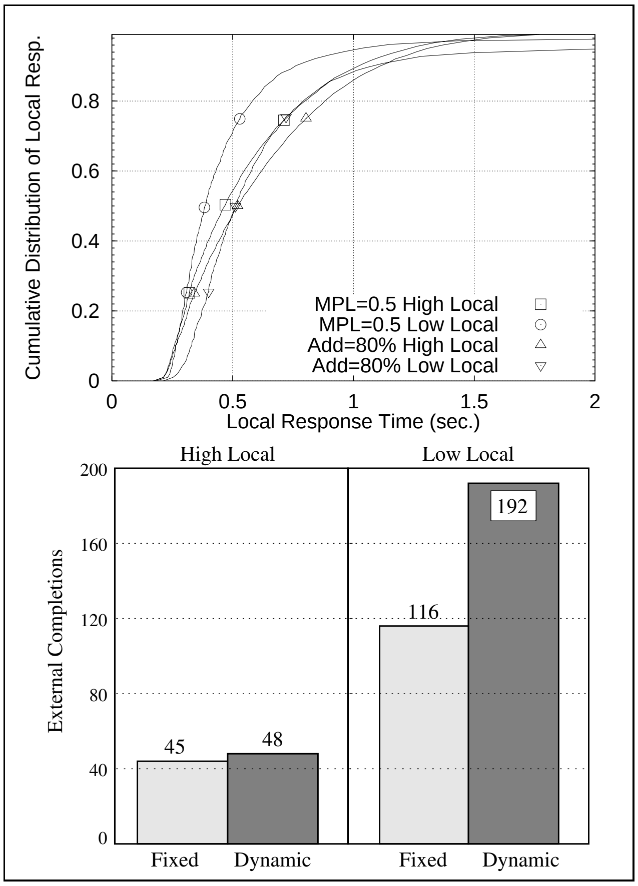

Figure 8. Dynamic vs. Fixed MPL

cal response time, therefore the techniques described still function properly in the presence of updates.

6.3. Experiment 3: Dynamic MPL

Figure 7 illustrates that by adjusting the add threshold, the utilization feedback can control the spacing of external requests much like fixed spacing. The advantage of this feedback mechanism is that it can aptly avoid periods when the local users place a heavy load on the system, or take advantage of periods of low activity. Figure 8 shows the difference between spacing and feedback. When we increased the average local interarrival time from 0.2 seconds to 0.4 seconds (i.e., decreased the local demand), the dynamic version kept the local users at the same QoS, but allowed more external work to be completed.

6.4. Experiment 4: Profiling

Profiling allows the PA to keep small jobs from waiting for large jobs to complete. This experiment, which is summarized in Table 2, illustrates the need for profiling by running an external workload with both large and small jobs. The arrival times of the queries were uniformly distributed over the 25 minute experiment. We assume that the administrator decided that they wanted at most one large query in the system at a time. Furthermore, they chose a dynamic MPL with an add utilization of 70%. In the first run, the profiler was not used, so small queries entered the same queue as the large queries. In the next run, the profiler was used, and the administrator decided that at most two small queries could enter the system at a time as long as the utilization was below 90%. So in this case, the PA scheduler consisted of two queues, one for small queries and one for large queries.

The results of the experiment are shown in Figure 9. The top chart shows that allowing the two small queries to execute along with the large query had little impact on the local QoS. The chart on the bottom, however, shows that the throughput of the small queries increased by 25 times when profiling was used, with only a small decrease (13%) in the throughput of the large queries.

Table 2. Mixed External Workload Parameters

Class	Parameter	Setting
large external	MPL	1
	add util.	70%
	#jobs	76
	queries	range count, 25-75% of relation with clustered index
small external	MPL	2
	add util.	90%
	#jobs	2530
	queries	point select with non-clustered index

7. Related Work

Our work inherits much from the work on operating system load control [11, 6]. Our contribution is in applying load control in a novel way to protect the local QoS. A significant amount of work on database scheduling has also been completed, especially on memory allocation [12, 10, 17, 18, 3, 5]. The main difference between our work and the database scheduling is that ours exists outside the database and is not specific to any particular DBMS.

A few vendors have some load control mechanisms similar to ours. IBM's DB2 for MVS [14] offers many tuning parameters, but it does not have our fractional MPL and profiling mechanisms. NCR's DBQM [20], which was developed concurrently with our research, includes many of the features described in this paper, but is specific to NCR's Teradata DBMS. In particular, DBQM uses both feedback and query profiles to schedule the external workload.

Two other classes of applications that extend the scheduling of DBMSs are available: Query Analyzers and Transaction Processing (TP) Monitors. Query analyzer products are sold, for example, by Platinum Technologies (Plan Analyzer, DB Analyzer, SQL-Spy, Detector) [21], Blue Lagoon (DBProfiler), and MicroStrategy (DSS Administrator) [19]. The analyzers provide a feature like our profiler to estimate the cost of executing a query before actually running it. The difference between query analyzers and our project is that the analyzers use this information to warn users and developers of poorly formed queries that could consume large amounts of resources, while we use the profile for scheduling. An exception to this is DSS Administrator which appears to use the profile information to schedule the queries from their Decision Support System (DSS) product.

TP Monitors are sold by Transarc (Encina [9]), BEA (Tuxedo, Top End [1, 22]), and IBM (CICS [13]), to name a few. The main thrust of a TP monitor is to coordinate transactions through a TP system, but TP monitors do offer priority queuing and load balancing [2]. The priorities are controlled by the applications, not through profiling, and their load balancing just tries to keep the machines in the TP system equally busy.

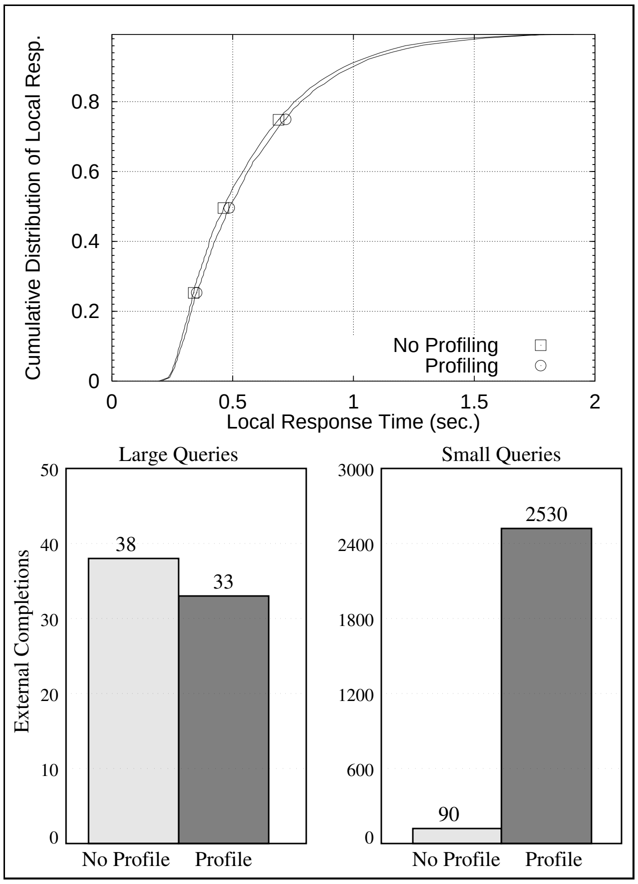

Figure 9. Profiling Results

8. Future Work

We have identified a number of areas for future work. We described some mechanisms in Section 4, for example the buffer manager, that we believe to be useful, but we have not yet demonstrated that fact with experiments. Another issue with our current prototype is that it can be unfair because small requests can keep large requests from starting. The grouping of multiple external requests into a transaction presents additional challenges to the PA from the perspective of locking and resource consumption.

9. Conclusions

We identified the potential performance dangers of allowing external access to an existing information system.

We believe organizations should address this problem before allowing external access. We offer the PA as a general solution that can, with little effort, protect most sites without requiring changes to the underlying system or any of the programs that the site uses for local access.

We described how load control, request profiling, buffering, and priority scheduling can be combined to form an elegant, novel solution. We demonstrated the need for load control, and that MPL and spacing effectively limit the impact of external requests, even in the presence of updates. Feedback is used to obtain a dynamic MPL that allows the PA to respond to changes in local workload. Profiling enables the administrator to base scheduling decisions on the resource requirements of a job. In particular, we showed that discriminating between large and small jobs improved the processing of the small requests by over 25 times while only marginally slowing the large requests. All these features working from the outside of the database combine to provide a realistic and effective solution to a very real problem.

We believe that the demand for the PA will continue to increase as Internet services, data mining, and data integration projects proliferate. We also believe that the ideas presented in this paper can be quickly integrated with existing products like middleware tools, TP monitors, multi-database systems, and database schedulers which implies that commercial products with PA-like features should appear in the near future.

Acknowledgements: The authors would like to thank Donko Donjerkovic for his work on the query profiler.

References

[1] J. M. Andrade. Open on-line transaction processing with the TUXEDO system. *IEEE CompCon*, 1992. also http://www.-beasys.com/.

[2] P. Bernstein. Transaction processing monitors. *Communications of the ACM*, 33(11), November 1990.

[3] K. P. Brown, M. Mehta, M. J. Carey, and M. Livny. Towards automated performance tuning for complex workloads. In *Proc. of the 20th VLDB Conf.*, Santiago, Chile, 1994.

[4] S. Chawathe et al. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. of IPSJ Conf.*, pages 7–18, Tokyo, Japan, October 1994.

[5] D. L. Davidson and G. Graefe. Dynamic resource brokering for multi-user query execution. In *SIGMOD 95*, pages 281–292, San Jose, CA, 1995. ACM.

[6] P. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, May 1968.

[7] W. Du, R. Krishnamurthy, and M.-C. Shan. Query optimization in heterogeneous dbms. In *Proc. of the 18th VLDB Conf.*, pages 277–291, 1992.

[8] D. Eichman. Ethical web agents. In *Electronic Proc. of the Second World Wide Web Conf. '94: Mosaic and the Web*, 1994. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/-Agents/eichmann.ethical/eichmann.html.

[9] J. L. Eppinger and S. Dietzen. Encina: Modular transaction processing. *IEEE CompCon*, 1992. also http://www.-transarc.com.

[10] C. Faloutsos, R. Ng, and T. Sellis. Predictive load control for flexible buffer allocation. In *Proc. of the 17th Int'l VLDB Conf.*, Barcelona, Spain, September 1991.

[11] D. Ferrari. *Computer Systems Performance Evaluation*. Prentice Hall, 1978.

[12] R. B. Hagmann and D. Ferrari. Performance analysis of several back-end database architectures. *ACM Transactions on Database Systems*, 11(1):1–26, March 1986.

[13] IBM Corp. CICS product family home page. http://www.-software.ibm.com/ts/cics/.

[14] IBM Corp. MVS workload manager / system resource manager. http://www.s390.ibm.com/products/mvs/wlm/-index.html.

[15] M. Koster. Guidelines for robot writers. http://info.-webcrawler.com/mak/projects/robots/guidelines.html.

[16] M. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *Proc. of the 8th Intl. Conf. of Distributed Computing Systems*, pages 104–111, June 1988.

[17] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. In *Proc. of the 19th VLDB Conf.*, Dublin, Ireland, 1993.

[18] M. Mehta, V. Soloviev, and D. J. DeWitt. Batch scheduling in parallel database systems. In *Proc. of the 9th Intl. Conf. on Data Engineering*, Vienna, Austria, April 1993.

[19] MicroStrategy. http://www.strategy.com/.

[20] NCR DBQM. http://www3.ncr.com/teradata/dbqm.pdf.

[21] Platinum Technologies. http://www.platinum.com/.

[22] R. Smerik. An overview of TOP END. *IEEE CompCon*, 1992.

[23] M. Stonebraker et al. Mariposa: a wide-area distributed system. *VLDB Journal*, pages 48–63, 1996.