Workload Class Importance Policy in Autonomic Database Management Systems

Harley Boughton
Queen's University
Kingston, ON, Canada
harley@cs.queensu.ca

Pat Martin
Queen's University
Kingston, ON, Canada
martin@cs.queensu.ca

Wendy Powley
Queen's University
Kingston, ON, Canada
wendy@cs.queensu.ca

Randy Horman
IBM Toronto Lab
Markham, ON, Canada
horman@ca.ibm.com

Abstract

A key advantage of Autonomic Computing Systems will be their ability to manage according to business policies. A key challenge to realizing this ability is the problem of automatically translating high-level business policies into low-level system tuning policies, which is the result of the different semantics used at the two levels.

Economic models, which are expressed using business level concepts, have been used successfully in computer resource allocation problems. In this paper, we utilize an economic model to map business policies to resource allocation decisions in a database management system (DBMS). We focus on business policies that describe the relative importance of competing workloads on a DBMS. We present experiments with a simulation of the model that investigate a number of meanings of importance and identify how this additional information can be used to effectively allocate main memory resources in a commercial DBMS.

1. Introduction

Computing system complexity is approaching a point where highly skilled system administrators, let alone managers with corporate policy decision making approval, are unable to comprehend all aspects of a system's day to day performance. This crisis has been brought to the attention of the computing world through initiatives such as IBM's Autonomic Computing [11]. Autonomic Computing systems should be self-configuring, self-tuning, self-protecting, and self-healing. The goal of the initiative is to enable the system to be managed more directly by business policies, which will allow decision makers to have direct control over the computing systems that are a central part of their business [11].

Database Management Systems (DBMSs) are a core component in most organizations' computing systems. DBMSs are so complex that many require specialized database administrators (DBAs) to be kept on staff, spending a large portion of time tuning the DBMS so that the system can meet its required throughput and response time goals. Most management policies, however, are written in terms of business-level concepts such as revenue and return on investment (ROI). DBAs must therefore attempt to translate business policies into low-level technical requirements for the DBMS. This is a non-trivial exercise as there is little similarity in the metrics used for measuring database performance and business performance [2].

One type of policy that has a great deal of impact on the way in which the DBA makes decisions is an "importance policy". This type of policy allows managers to differentiate the importance of work being done on the DBMS, which in turn can provide the DBA with additional information that can be used in making configuration decisions. Importance becomes more critical as businesses consolidate workloads of different business units onto a single DBMS.

The buffer area, for example, is a region of main memory reserved by a DBMS for caching data pages to reduce disk accesses. It is a critical factor in database performance [4]. The buffer area is typically partitioned into independent buffer pools and page replacement is carried out separately within each buffer pool. Determining the sizes of the buffer pools is usually done by first making an initial estimate based on workload and database characterizations and then refining the estimate by monitoring a number of buffer pool parameters, most notably buffer hit rate. The hit rate is the percentage of the time when a requested page of information, either an index page or data page, is found in the buffer cache. This is a common measure of the performance of a single buffer pool. Tian et al [14] attempt to automate this process. Additional importance information can assist in making a proper decision on how to determine these sizes.

A key problem in implementing an importance policy is determining what it means to say that one workload class is more important than another. When utilized to make tuning decisions, different interpretations of importance are possible. For example, a more important class can get access to the resources it needs before a less important class; an important class can hold resources not in use in anticipation of work to come, or an important class can appropriate resources from less important classes when needed. Another problem that needs to be addressed when implementing an importance policy is the degree to which the relative level of importance of classes affects tuning decisions.

One technique that can be used to address the disparity between high-level and low-level metrics is to introduce an economic model into the low-level system. Economic models have been used in a number of resource allocation problems in computing with great success [6][8][13][19]. The models are easily understood by many of the business policy makers and do not require the specialized IT knowledge that other models require. This eases the translation as the low-level system now functions similar to many high-level business policies. This allows for a system that can be managed directly according to business policies.

The overall goal of our research is to facilitate the evolution towards "business policy based" autonomic tuning. The main contribution of the paper is an investigation of how workload class importance should affect low-level resource allocation. Given a workload that has been segregated into classes, with each class having an associated importance level, we look at a number of ways in which this added information can affect resource allocation decisions in an autonomic database management system. We create an economic model representation of the buffer pool sizing problem and implement the model as an offline simulation. The model is utilized to investigate a number of possible meanings for importance and to identify how this information can be used to allocate buffer pool memory between competing workloads. We present experimental results where we look at a three class workload in which the classes are competing for buffer pool memory. We use the simulation to make memory allocation decisions for these workloads according to their level of importance and estimated need for buffer pool space.

The remainder of this paper is structured as follows. Section 2 discusses related background work. Section 3 details the economic model utilized in the simulation. Section 4 discusses the various definitions of importance used in the experiments. Section 5 presents our experimental results. Section 6 presents our conclusions and guidelines for future work.

2. Background and Related Work

This research draws from a number of areas in order to address the problem of implementing business policy based self-tuning in a Database Management System. This section provides some background information and references previous research in each of the four main areas addressed by this work.

2.1 Autonomic Database Management Systems

Since 2001, when IBM introduced their Autonomic Computing Manifesto [11], there has been great interest in Autonomic Computing within the scientific community. Great strides have been made on a number of issues key to developing the types of self-configuring, self-tuning, self-protecting, and self-healing systems outlined. Research on self-tuning DBMSs has included such topics as index selection [16], materialized view selection [1] and memory management [4][20].

In reference to Ganek's and Corbi's evolution towards autonomic operation [9], however, much of this work fits into the Adaptive Computing level. The key feature that differentiates Autonomic Computing from Adaptive Computing is the ability to manage according to business policies. Whereas the goals used in much of the previous work involve IT metrics such as response time or throughput [4][14], there is little work that directly addresses the issue of managing IT systems according to high-level business policies [2]. Our work is concerned with the implementation of business policies in an attempt to achieve the Autonomic Computing level.

2.2 Goal-Oriented Resource Allocation

Historically, much of the work done in regards to DBMS resource allocation involved optimizing system wide performance, while more recent work separates the workload into classes that share some commonality [4]. Many solutions involve segregating portions of the workload and allocating resources to each class as the resource requirements for a class can be better understood. Instead of trying to optimize overall system allocations, a DBA can optimize the allocation of each class and achieve an overall system optimization [4].

One of the most important resources in determining system performance is memory management [4]. There are a number of parameters that a DBA must tune when optimizing memory usage, but one of the most common is determining the size of the buffer area. The buffer pool sizing problem involves finding the optimal allocations of memory for a set of buffer pools so that the best possible database performance is achieved [14]. Previous work, however, examines this problem from the perspective of

the database objects as opposed to multi-class workloads with separate buffer pools assigned to workload classes.

We examine the buffer pool sizing problem in the case where multiple workload classes run concurrently on the same DBMS, with a separate buffer pool assigned to each class. Our research further builds upon the class model by introducing importance information into the resource allocation problem. As opposed to attempting to optimize each class to an equal degree, we use class importance to give priority to optimizing performance for some classes more than others.

2.3 Economies for Resource Allocation

Many researchers have found economic models bring benefits to the resource allocation problem in computing systems [6][8][13]. The benefits include utility functions to describe agent allocation preferences in a concise mathematical formula [18], decentralization through the use of multiple brokers, agents and arbitrageurs, and scalable resource allocation solutions [8].

Utility, as used in economics, describes an amount of happiness or satisfaction gained from consumption of an allocation of a commodity. A utility function is a mapping of this satisfaction to various allocations of the commodity. In terms of computing resources, utility can be thought of as a measure of usefulness and a utility function provides a mapping between all allocations of that resource and the usefulness achieved. Even with this very understandable application to computer resource allocation problems, the study of the practical applications of utility functions in computing is still fairly new [18].

Economic models typically involve suppliers and consumers that exchange goods. There are many different ways to implement this, with each implementation providing certain features to accomplish a specific goal. Using wealth and auctions creates a transparent decision making process for allocations, while direct trading of resources between agents will provide an efficient convergence to a Pareto optimal resource allocation [19]. Pareto optimality implies that no individual's utility can be improved without diminishing the utility of others. This is a very desirable quality for the solutions provided by the economic model.

2.4. Business Policy Management and Priority

Although researchers have been pursuing Autonomic Computing for a number of years, there is very little research that examines self-tuning systems guided by business policies [2]. Most research is concerned with optimizing traditional IT metrics.

One business policy of particular interest is an importance policy, which is useful in organizations utilizing management ideas such as Strategic Business Unit (SBU) valuation that comes from Value Based Management [17]. By determining a valuation for an SBU, managers are able to determine the appropriate SBU to which new capital should be assigned. In a situation where computing infrastructure is shared among these business units, one can readily see how an importance policy could translate to priority for computing resources.

In relation to DBMSs, much of the work on using importance/priority information has focused on scheduling queries [5]. Previous research on using priority to manage physical resources in a DBMS has attempted to transfer ideas of priority from other computing resources such as CPU scheduling and buffer management [5]. However, there are a number of issues not addressed, such as the degree of difference in importance between various levels of priority and what priority should mean in resource allocation.

Previous work in utilizing priority information has focused on "real-time database systems" (RDBMS) [7]. Priority information is typically used as additional information for making different decisions, such as CPU scheduling or concurrency conflict resolution. However, RDBMSs are typically interested in allocating resources for individual queries to meet specific deadlines as opposed to adjusting for class-based performance goals. Additionally, most priority schemes are based on adjusting schedules through some sort of admission control policy.

3. Economic Model

The economic model shown in Figure 1 solves a version of the buffer pool sizing problem. We consider the case in which a DBMS concurrently runs several *workloads*, where a workload corresponds to a related set of the requests and data objects, for example from a particular application. We assume that each workload on the DBMS is assigned its own buffer pool. This organization can be generalized to a separate resource partition in the DBMS for each workload.

In this paper, we utilize an instance of the model consisting of three consumers and a single broker. The broker is responsible for allocating buffer pool space to the consumers. It sells blocks of memory pages to consumers using sealed-bid auctions.

The consumers are agents representing three workload classes running simultaneously on the DBMS. We conceptually divide each workload into a series of segments of approximately equal numbers of requests and perform a new reallocation of resources at the beginning of each segment. The class agents are assigned an amount of wealth based on an estimate of the work they must complete in a segment and on the relative importance of the workload.

The class agents employ a utility function to determine the maximum amount of wealth they are willing to spend for the block of memory pages currently available for auction. The agents submit this value as a sealed bid to the resource broker who selects the highest bid as the winner and assigns the resource accordingly. This continues until all resources have been allocated or until no class agent desires more resources.

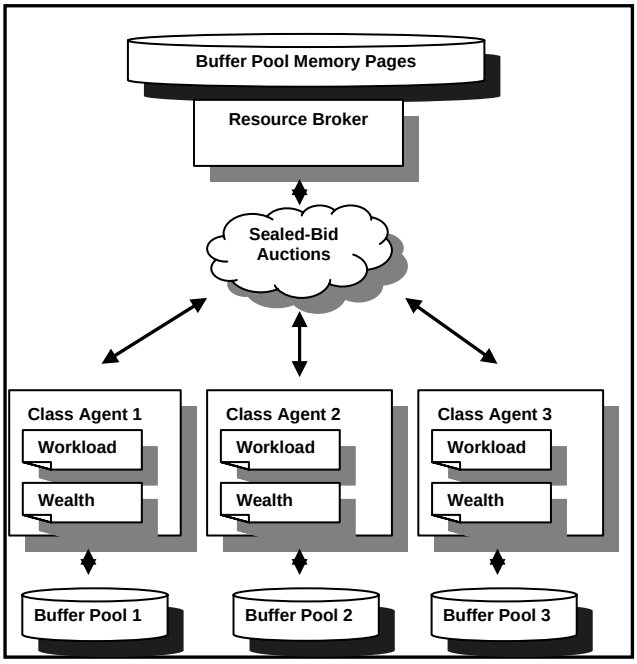

Figure 1: Economic model

A class agent's utility curve represents the usefulness of an allocation of buffer pool memory to the agent. Hit rate is the most commonly used metric to evaluate buffer pool performance and directly depends upon the size of the buffer pool. We therefore adopt a well-known hit rate estimation function, known as Belady's equation [3], as the utility function for our class agents.

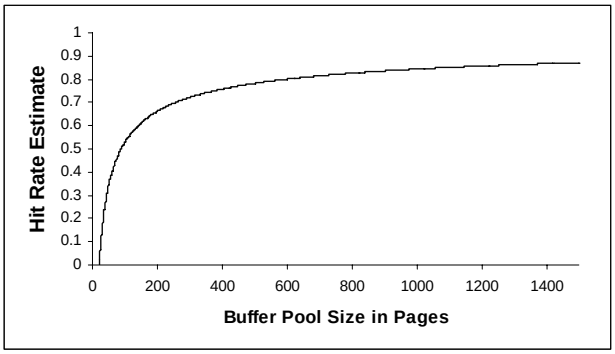

Figure 2: Sample curve formed by Belady's equation

As shown in Figure 2, Belady's equation allows us to use the marginal increase in hit rate as the marginal utility for a given increase of buffer pool memory. This monotonic decreasing utility curve in a competitive model also allows us to achieve a Pareto-optimal allocation [18]. As an agent wins auctions, it gains resources, but loses wealth. It is, therefore, less likely to bid on additional resources as it gives up wealth and thus allows other classes with less wealth and resources the chance to win resources. Agents make bids until they have sufficient resources, insufficient wealth, or all resources have been claimed.

4. Importance Policies

Our research addresses two key problems in translating an importance policy into tuning policies in an autonomic DBMS. The first problem is determining the degree of importance, that is, how much more important a high-importance class is compared to a low-importance class. The second problem is how to best define importance in terms of how resources are allocated.

We consider here a scenario that involves assigning one of three importance levels to each workload class running on the DBMS. We use the labels "High Importance", "Normal Importance", and "Best Effort". A class that is "High Importance" should demonstrate some priority over classes that have "Normal Importance" or "Best Effort" importance. In a consolidated enterprise system, this "High Importance" label could, for example, be applied to a class of work representing an OLTP-like order entry department as this work is directly revenue generating, or to a class of work corresponding to queries entered by the company CEO and thus considered most important. The "Normal Importance" label would likely apply to all other business related workloads such as an HR department that generates reports that they need during business hours, but have a lower priority than business units directly affecting revenue. Finally, the "Best Effort" label would be applied to classes of transactions that do not have any strict deadline, such as background DBMS maintenance work, or after hours reporting queries.

These three levels, we believe, provide a reasonable scenario for implementing importance in the DBMS. These three labels allow, in addition to normal workloads, a way to both raise and lower the importance of workloads on the DBMS. A discrete labeling system is also preferred to a continuous value describing importance as it is much easier to make decisions choosing the appropriate level for a class from a small number of well-defined importance levels.

4.1. Degree of Importance

A key problem in implementing different levels of importance is how to differentiate between the levels. This involves determining how much more important one level is than another. The values are key when using the importance levels in low-level resource allocation decisions. In this work, we determine appropriate weights for the importance levels through experimentation with the economic model.

Each consumer agent in the economic model is responsible for a workload. An estimate of the amount of work to complete (obtained in our experiments by using the sum of the estimated number of I/O operations per query from the IBM DB2® Universal Database™ [10] Explain utility) is used to determine the wealth awarded to the agent for the current allocation interval. This is then multiplied by an "importance multiplier" to enforce a degree of importance. For example, an agent with a "High" degree of importance may have a multiplier of 3.0, while an agent with a "Best Effort" degree of importance may have a multiplier of only 1.0. By adjusting these multipliers, we affect the amount that one class is more important than another by altering their ability to outbid other classes for resources. In our experiments, we look at the impact of a range of values for these multipliers.

4.2. Definition of Importance

Paramount in implementing importance in the DBMS is defining what importance means. This definition describes the differences in entitlements and abilities between a high-priority class and a low-priority class.

For this work, we are concerned with defining what importance means with regard to resource allocation. We therefore base our definition on the entitlement one class has to resources as compared to other classes. We present three definitions of importance and experiment with these using our economic model simulation. These definitions of importance represent two classic definitions of priority, namely Non-Preemptive and Preemptive priority [5] as well as a variation on the Preemptive model where High Importance classes are exempt from preemption.

The first definition of importance states that all classes are entitled to their minimum necessary resource allocation. Additional resources available in the system are more likely to be assigned to important classes. This represents a Non-Preemptive model. All classes are able to complete some work during an interval, with any additional resources being used to improve the performance of important work. This allows us to provide a minimum guaranteed level of performance. In our economic model, this is accomplished by pre-allocating the minimum required resources to each class before the auctions begin. However, if the total resources available are less than the sum of the minimum allocations, we then scale the allocations according to their ratio of the total resources requested. To calculate the minimum required resources, we pick a target buffer pool performance level and use the class's utility curve to calculate the necessary resource allocation to meet that hit rate.

The second definition of importance states that the requirements of important classes should be satisfied before those of less important classes. Important classes may be allocated all resources in the system such that less important classes must wait until resources are made available. This definition represents a preemptive priority model. For the economic model simulation, this means that all resources are auctioned to the highest bidding class through the competitive mechanism.

The final definition of importance guarantees a high level of performance to High Importance classes and allows other classes to compete for remaining resources. The purpose of this importance scheme is to address the possibility of a lower-importance class preempting the High Importance class. Similar to the Non-Preemptive scheme, we implement this in the economic model by a pre-allocation of resources to the High Importance class. The pre-allocated resource amount is based on a high level of performance for the class buffer pool.

4.3. Aging Importance

Allowing priorities to age is another interesting aspect of an importance policy that we examine in this work. Aging involves gradually incrementing the priority of objects over time. It is particularly relevant to our preemptive-style importance policy definition and allows a lower importance class to eventually overcome the preemption of the High Importance class.

In our approach, aging involves allowing a class to better compete for resources the longer the workload has been running. To implement this in the economic model, we allow classes to accumulate the wealth they do not spend in previous allocation periods. Thus, if a class is preempted, it will have additional wealth in the next allocation period and be better able to compete against other classes that have likely expended all their wealth in the previous period. For the non-aging schemes, classes do not carry over any wealth that is unspent.

5. Experimental Results

The experimental environment is IBM's DB2 Universal Database (DB2 UDB) version 8.2 [10] running on an IBM xSeries 240 PC server with the Windows XP operating system. The server is equipped with two 1 GHz

Pentium 3 processors, 2 GB of RAM and an array of 22 disks.

We use a single instance of the DBMS with three identical databases. We input three OLTP workload scripts, each consisting of 120,000 requests from the 5 different transactions of the TPC-C benchmark [15]. The sets of transactions in the workloads are the same, but the order and proportions of the transactions vary, which provides workloads with different resource needs. The workloads are divided into 12 segments of 10,000 queries. The start of each segment provides a checkpoint at which we resize the buffer pools during a run. The three workloads simulate a consolidated workload reminiscent of a single organization with three business units running separate OLTP workloads simultaneously against different sets of tables within the same database.

The workloads are entered as input to the economic model simulator, which produces a list of allocations at each of the checkpoints. At each segment checkpoint in the workload script, the buffer pool sizes are modified using the ALTER BUFFERPOOL statement with the IMMEDIATE keyword, which allows for dynamic resizing of the buffer pools without restarting the DBMS. When the command is used to increase the size of a buffer pool, additional memory pages are simply added from the database shared memory. If the size is decreased, pages are released from the LRU (least recently used) queue [12]. The dynamic resizing of the buffer pool selects the best candidate pages to be released first in order to minimize disk accesses. We assume that DB2 UDB dynamically resizes the buffer pools with the least possible impact.

Once the allocations have been determined by the simulation, they are entered in the workload scripts and the databases are restored to their initial states. The workloads are then run concurrently while we monitor the buffer pools throughout execution and collect statistics at each segment checkpoint. We record the number of logical and physical reads for data and indexes for each of the three buffer pools. This data is used to calculate the hit rates for each segment of the workloads. The values presented are the average of three runs of each set of scripts.

5.1. Degree of Importance

A proper set of importance multipliers within an economic model provides the highest price difference between what the high priority class is paying for resources compared to the lower priority classes. The large difference demonstrates that the high priority classes are able to purchase the resources needed before the lower priority classes. However, this is mitigated by the need to cause the least impact to the lower priority classes so that overall system performance is impacted as little as possible. Since hit rate is typically proportional to buffer space allocation, we look for the highest average allocation for the normal priority and Best Effort classes. Thus, we look for the best combination of the following two metrics:

- Price Difference: the difference in average price paid for resources between high priority and Best Effort classes, a higher value is better.
- Average Allocation: The average number of buffer pool pages allocated to the normal priority and Best Effort classes, a higher value is better.

The results for each metric are normalized and then combined into a single metric. Finally, we select the two best candidates and measure their performance on the test setup. We record the number of physical reads generated by the buffer pool configurations suggested by the economic model simulator using each of the multiplier sets. A physical read occurs when the workload requires a page that is not available in the buffer cache. Therefore, the lower the number of physical reads required, the better the overall system performance.

We tried five different sets of multipliers. In order of Best Effort, Normal, and High Importance, the sets are: (1.0, 2.0, 3.0), (1.0, 2.0, 5.0), (1.0, 2.0, 10.0), (1.0, 5.0, 6.0), and (1.0, 5.0, 10.0). These represent a number of different relative differences in importance, such as High Importance being similar to Best Effort or very much preferred, and Normal Importance spanning the range from Best Effort to High Importance. We test these parameter settings using the Preemptive model of importance with and without aging.

The results of the simulation for these different sets of multipliers, as shown in Figure 3, provided two sets that performed better than the others according to our criteria of combined price difference and average allocation.

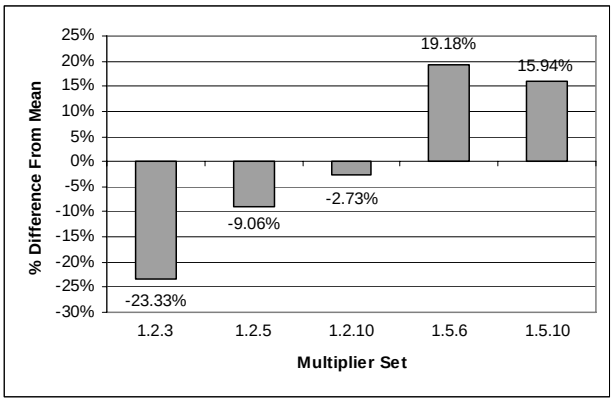

Figure 3: Combined normalized price difference and average allocation rank

Figure 4 compares the physical reads generated from the two best sets of multipliers. We see that the set (1.0,

5.0, 6.0) resulted in 5.55% fewer physical reads than (1.0, 5.0, 10.0). Thus, we selected (1.0, 5.0, 6.0) as our multiplier set for our further experiments in implementing an importance policy. Although these multipliers may be specific to these experiments, there are rules-of-thumb that can be observed from these results:

- The Best Effort class should have a multiplier of one to ensure that cannot be allocated resources at anything but the minimum price.

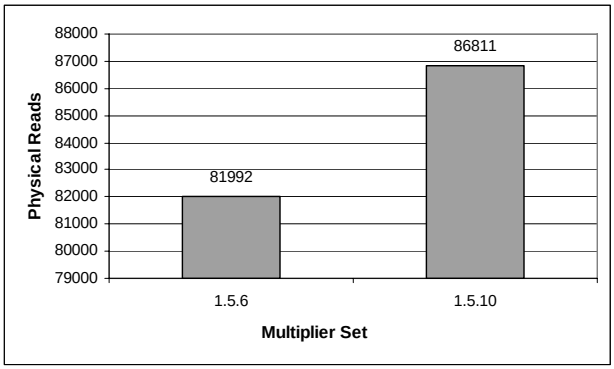

Figure 4: Total physical reads on test system using best candidate multiplier sets

- The High Importance class should have a significantly higher multiplier than the Best Effort class to ensure that it has the wealth to purchase resources and reach its desired allocation before other classes.
- The Normal Importance class should have a multiplier similar to that of the High Importance class so that its overall impact on the system is mitigated by a higher average allocation, resulting in a lower total number of physical reads.

5.2. Definition of Importance

We investigated the impact of the definitions of importance explained in Section 4. These include:

- Preemptive: In a Preemptive scheme, one class may appropriate all resources preempting the execution of others.
- Non-Preemptive: In the Non-Preemptive scheme, a minimum amount of resources are guaranteed to all classes.
- Preemptive Exempting High Importance: In this scheme, we pre-allocate the resources the high priority class needs and use a preemptive model for the remaining resources. This guarantees a level of performance only for important classes.

To evaluate the effect of the importance definitions, we examine two criteria. First, we want an importance scheme that provides the highest benefit to the classes designated as High Importance. The benefit is evident by the hit rate that is achieved from the allocations provided by the simulation.

Secondly, we want to provide this benefit to important classes with as little effect on the rest of the system as possible. Thus, we also look at the total number of physical reads recorded by the system for each importance scheme. Again, a lower score is better for total physical reads. We claim that the scheme that can provide the best combination of these two criteria provides us the most beneficial definition of importance when looking at allocating buffer pool space.

The two importance schemes that offer an initial allocation of memory also required additional experimentation to determine the appropriate initial allocation. For the Non-Preemptive schemes, we looked for a minimum value that would allow every class agent to achieve a modest buffer pool hit rate. We tried initial allocations targeting a 50% and a 75% buffer pool hit rate for each class agent. The allocation is determined using the inverse of Belady's equation for each class agent. This is meant to act as a guaranteed minimum level of performance for all workload classes.

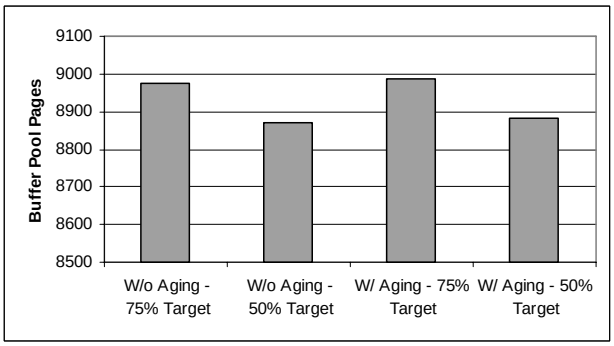

Figure 5: Average allocation of Normal and Best Effort classes using target 50% and 75% buffer pool hit rate initial allocations for Non-Preemptive scheme

We found little effect of the pre-allocated resources on the hit rate achieved by the High Importance class. This is due to the High Importance class agent's ability to acquire additional resources. However, there is a noticeable difference in the average allocations of the Normal and Best Effort classes as shown in Figure 5. Using an initial allocation targeting 75% hit rate results in a higher average allocation for the two lower importance classes. The higher average allocations in turn result in a lower number of total physical reads due to the diminishing returns nature of the relationship between buffer pool size and hit rate. By reallocating memory from the High Importance class to the less important classes, the increase in physical reads for the High

Importance class is more than offset by the decrease in physical reads for the lower importance classes. Thus, we use a 75% target for the initial allocation in the following experiments.

For the High Importance Exempt Preemptive scheme, we tried to provide a high level of performance to the High Importance classes while still allowing some resources to be available to less important classes. We selected initial allocations targeting 80%, 90% and 95% buffer pool hit rates for the High Importance classes while the less important classes receive no initial allocation. The High Importance classes therefore do not have their performance impacted by a lower importance class with a large amount of wealth.

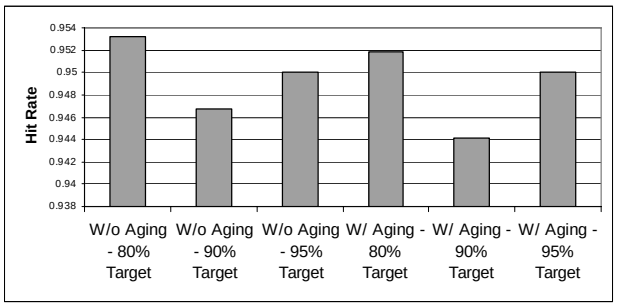

Figure 6: High Importance class hit rates achieved using target 80%, 90%, and 95% buffer pool hit rate initial allocations for Preemptive Exempt High Importance scheme

As shown in Figure 6, we see the best hit rates are achieved by the 80% and 95% allocations. However, when looking at the allocations received by the High Importance class under each of these schemes, we see that under the 80% scheme the class must still compete for most of its allocation (Figure 7) to achieve the high hit rate. This is not a desired quality as another class could acquire those resources.

The target 95% buffer pool hit rate initial allocation scheme gives a High Importance class the resources it needs so that it can maintain a very high hit rate without needing to compete for resources. This is the quality of solution we require, so the target 95% buffer pool hit rate initial allocation for the High Importance class is used in all further experiments.

Finally, we compare the importance schemes to find the one that provides the best combination of the metrics presented. We ran the simulation using each of the six schemes presented. We then ran the resulting allocations on our test setup three times, recording both hit rates and physical reads.

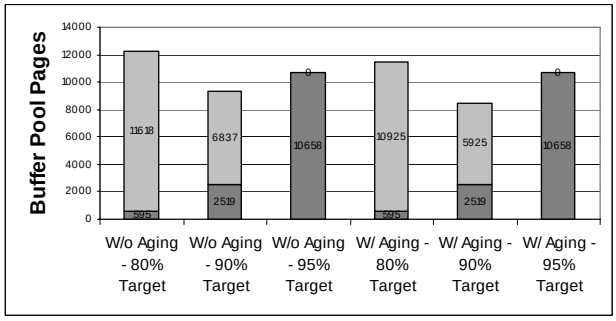

Figure 7: Initial allocation versus total allocation under each scheme (darker portion is initial allocation, lighter portion represents resources won through competition)

We first examine the difference between schemes with and without aging. Figure 8 shows that, in general, there a slightly lower hit rate is achieved for the High Importance classes when aging is involved. We also see a lower total number of physical reads in schemes that include aging.

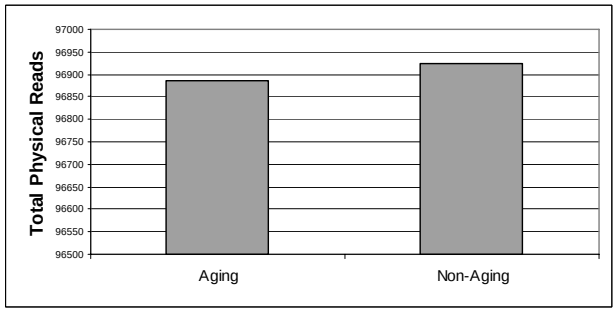

Figure 8: Comparing average physical reads for Importance schemes with and without aging

There is no obviously better scheme, however, the results of implementing aging do seem to create a slight moderating effect, as should be expected. Aging was implemented to allow lower importance classes to eventually be more competitive. In some intervals the lower importance class agents will win resources away from the High Importance class. Due to the diminishing returns nature of the relationship between buffer pool size and buffer pool hit rate, this will lower the hit rate of the High Importance class. However, the increase in physical reads incurred by the High Importance class will be more than offset by the decrease in physical reads for the other classes as their hit rates increase.

Figure 9, shows each of the different importance schemes normalized and combined scores in the two metrics. Since we want a higher hit rate for the High Importance class and a lower total number of physical reads across all classes, a combined score closest to zero provides the best balance of the criteria.

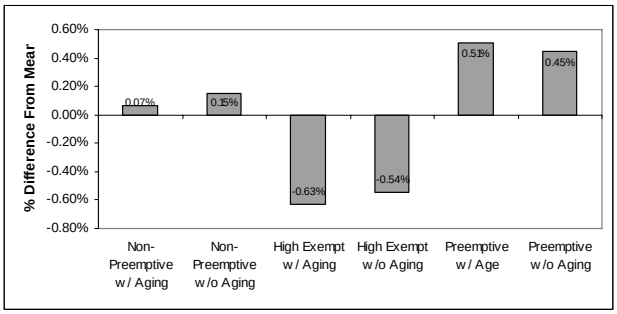

Figure 9: Combined normalized High Importance Class hit rate and total physical reads ranking

When looking at each scheme and their performance when we normalize the metrics, we see that the Non-Preemptive schemes provide the best balance of the two criteria. Semantically, this scheme would indicate that all workload classes can be guaranteed a minimum level of performance, while High Importance classes will be able to reach a significantly higher level and overall system performance is impacted as little as possible. Through a basic economic model, this scheme can be achieved in an automated fashion through simple rules of wealth assignment and trade.

6. Conclusions and Future Work

In this paper we present our initial research on the problem of facilitating "business policy based" autonomic tuning. The main contributions of the paper are an approach to mapping business policies to system-level tuning actions based on an economic model and an investigation, using the approach, of how workload class importance should affect low-level resource allocation.

Owing to the use of our economic model for resource allocation, through minor rule and parameter changes, we are able to implement a variety of possible importance policies. We found that the use of an economic model greatly eases the implementation of a Workload Class Importance Policy. The added benefits of decentralization, automation, and simple consumer agents make the economic model an interesting approach to resource allocation.

We implemented a simulator of our economic model and used the simulator to investigate the impact of three aspects of importance on system performance. We first examined ways to define the relative importance of competing workload classes. We found that, for our scenario, High Importance classes should have a much higher degree of importance than Best Effort classes and that Normal Importance classes should have a degree of importance close to the High Importance classes. This organization balanced the need to favour High Importance classes with achieving good overall system performance.

Second, we found that, for our scenario, the Non-Preemptive definitions of importance provide the best balance of a high hit rate for important workloads and good overall performance. The preemptive definitions allocate pages to High Importance classes at the expense of the other classes, which means the total number of physical reads becomes larger. The Preemptive Except High Importance definitions actually result in lower hit rates for the High Importance class than the Non-Preemptive schemes. In the Preemptive Except High Importance schemes, the High Importance class is given an initial allocation that provides a guaranteed minimum hit rate. The High Importance class must give up a corresponding amount of wealth for the initial allocation and so is not able to compete to acquire a sufficient amount of additional buffer pages. The experiments indicate that all workloads can be allocated enough memory resources to guarantee a minimum level of performance, while High Importance workloads will still be able to reach a higher level of performance. A basic economic model can easily achieve this preferred situation in an automated fashion through simple rules of wealth assignment and trade.

Third, we found that, for our scenario, aging importance schemes consistently give a lower average hit rate for the High Importance class compared to the same importance scheme without aging, while resulting in a lower number of physical reads system wide. The schemes using aging allow the lower importance classes to acquire some of the resources from the higher importance classes on occasion. This is again due to the relationship between buffer pool size and hit rate.

There are a number of interesting avenues of future research suggested by this work. We can consider scenarios involving other types of workloads such as OLAP or electronic commerce. Hit rates for these other types of workloads behave in much the same way as those for OLTP workloads so we can expect similar results. We can extend the economic model to allocate multiple resources by adding additional brokers and to allow classes to trade-off between the various resources by utilizing multiple utility functions. Additionally, more economic features, such as a futures market, could be added to allow agents to choose when to execute to maximize performance for a given budget. This would be especially useful in a system where the wealth of a class represents a real-world budget.

Implementing additional business policies such as an Operating Costs policy where the broker can only lend out resources in accordance with the costs it incurs for the resource to run may be useful in guiding systems towards profit maximization.

We believe economic models are a promising approach to implementing policy-based tuning in autonomic DBMSs. There are, however, a number of challenges to be faced in realizing this approach in production systems: (1) identify and maintain utility functions for various resources for different types of workloads; (2) deciding on effective points to reallocate resources during execution; (3) predicting the upcoming requests at these points in order to determine wealth for each workload class, and (4) integrating the decision making into the DBMS engine for efficiency.

Acknowledgements

The authors gratefully acknowledge financial support from IBM Canada Ltd. and Communications and Information Technology Ontario (CITO).

References

[1] S. Agrawal., S. Chaudhuri, and V. Narasayya, "Automated Selection of Materialized Views and Indexes," *Proceedings of the 2000 International Conference on Very Large Databases*, Cairo, Egypt, 2000.

[2] S. Aiber, D. Gilat, A. Landau, N. Razinkov, A. Sela, and S. Wasserkrug, "Autonomic Self-Optimization According to Business Objectives". *Proceedings of the International Conference on Autonomic Computing*, 2004, pp. 206-213.

[3] L. Belady, "A Study of Replacement Algorithms for Virtual Storage". Computer, *IBM System Journal*, 5(2), July, 1966, pp. 78-101.

[4] K.P. Brown, M.J. Carey, and M. Livny, "Goal-Oriented Buffer Management Revisited", *Proceedings of the 1996 Association for Computing Machinery Special Interest Group on Management of Data*, 1996, pp. 353-364.

[5] M.J. Carey, R. Jauhari, and M. Livny, "Priority in DBMS Resource Scheduling". *Proceedings of the 1989 International Conference on Very Large Databases*, 1989 pp. 397-410.

[6] G. Cheliotis, and C. Kenyon, "Autonomic Economics: Why Self-Managed e-Business Systems Will Talk Money". *IEEE International Conference on E-Commerce*, 2003, pp. 120-127.

[7] A. Datta, S.H. Son, and V. Kumar, "Is a Bird in the Hand Worth More than Two in the Bush? Limitations of Priority Cognizance in Conflict Resolution for Firm Real-Time Database Systems". *IEEE Transactions on Computers*, 49(5), 1999, pp. 482-502.

[8] D.F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, "Economic Models for Allocating Resources in Computer Systems". In Scott Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*, Scott Clearwater. World Scientific, Hong Kong, 1996.

[9] A.G. Ganek, and T.A. Corbi, "The Dawning of the Autonomic Computing Era". *IBM Systems Journal* 42(1), 2003, pp. 5 – 18.

[10] IBM. "IBM Software – DB2 Universal Database for Linux, UNIX and Windows – Product Overview". Retrieved October 30, 2005 from http://www-306.ibm.com/software/data/db2/udb/

[11] J.O. Kephart, and D.M. Chess, "The Vision of Autonomic Computing". *IEEE Computer*, 36(1), 2003, pp. 41–52.

[13] N. Stratford, and R. Mortier, "An Economic Approach to Adaptive Resource Management". *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*, 1999, pp. 142-148.

[14] W. Tian, P. Martin, and W. Powley, "Techniques for Automatically Sizing Multiple Buffer Pools in DB2". *Proceedings of Center of Advanced Studies Conference (CASCON)*, Toronto, Canada, 2003, pp. 294-302.

[15] Transaction Processing Performance Council. "TPC-C". 2005, Retrieved December 1, 2005 from http://www.tpc.org/tpcc/default.asp

[16] G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelly, "DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes," *Proceedings of International Conference on Data Engineering*, San Diego, California, 2000, pp. 101-110.

[17] Value Based Management.net, "What is Value Based Management". Retrieved September 5, 2005, from http://www.valuebasedmanagement.net.

[18] W.E. Walsh, G. Tesauro, J.O. Kephart, and R. Das, "Utility Functions in Autonomic Systems." *Proceedings of the International Conference on Autonomic Computing*, 2004, pp. 70-77.

[19] M.P. Wellman, (1996). "Market-Oriented Programming: Some Early Lessons". *Market-Based Control: A Paradigm for Distributed Resource Allocation*, Scott Clearwater, World Scientific, River Edge, New Jersey, 1996.

[20] X. Xu, P. Martin, and W. Powley, "Configuring Buffer Pools in DB2 UDB", *Proceedings of Center of Advanced Studies Conference (CASCON)*. Toronto, Ontario, Canada, 2002.