2010

System identification in automatic database memory tuning

Tiffany Burrell
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd
Part of the American Studies Commons

Scholar Commons Citation

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
To Jehovah and my lovely parents
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>CHAPTER 1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER 2 BACKGROUND</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Optimal Access Paths</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Buffer Management</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Control Theory</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER 3 PLATFORM</td>
<td>16</td>
</tr>
<tr>
<td>3.1 Modifications of Database Management System</td>
<td>17</td>
</tr>
<tr>
<td>3.2 Actuator</td>
<td>19</td>
</tr>
<tr>
<td>CHAPTER 4 EXPERIMENTAL RESULTS</td>
<td>22</td>
</tr>
<tr>
<td>4.1 Workload</td>
<td>22</td>
</tr>
<tr>
<td>4.2 System Identification</td>
<td>22</td>
</tr>
<tr>
<td>4.3 Results</td>
<td>24</td>
</tr>
<tr>
<td>CHAPTER 5 CONCLUSIONS</td>
<td>29</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>30</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1  Initial Processes in Buffer Pool  2
Table 2  Processes in Buffer Pool Ultimately  2
Table 3  TPC-H Benchmark Queries Utilized  22
Table 4  List of Step Function Tests  23
LIST OF FIGURES

Figure 1  State of Initial Buffer Pool  2
Figure 2  State of Buffer Pool Ultimately  2
Figure 3  Workload Whose Intensity Increases as a Step Function of Time  12
Figure 4  Dynamic Workload  12
Figure 5  Block Diagram of Open Loop Control System  13
Figure 6  Block Diagram of Closed Loop Control System  14
Figure 7  Platform  18
Figure 8  Test 1 MPL Goes from 10 to 15  26
Figure 9  Test 1 Second Order Model  26
Figure 10  Test 2 MPL Goes from 10 to 20  26
Figure 11  Test 3 MPL Goes from 10 to 25  27
Figure 12  Test 4 MPL Goes from 10 to 30  27
Figure 13  Test 5 MPL Goes from 25 to 50  27
Figure 14  Test 6 MPL Goes from 25 to 80  28
System Identification in Automatic Database Memory Tuning

Tiffany Burrell

ABSTRACT

Databases are very complex systems that require database system administrators to perform system tuning in order to achieve optimal performance. Memory tuning is vital to the performance of a database system because when the database workload exceeds its memory capacity, the results of the queries running on a system are delayed and can cause substantial user dissatisfaction. In order to solve this problem, this thesis presents a platform modeled after a closed control feedback loop to control the level of multi-query processing. Utilizing this platform provides two key assets. First, the system identification is acquired, which is one of two crucial steps involved in developing a closed feedback loop. Second, the platform provides a means to experimentally study database tuning problem and verify the effectiveness of research ideas related to database performance.
CHAPTER 1

INTRODUCTION

Database systems are used in hospitals, banks, major corporations, and many other areas that require the collection of information. Even though a database is “just a database” to most of its frontend users, the venue to which a database belongs determines its requirement analysis, physical database design, hardware capacity, settings for runtime resource management, and the management of inter-system dependencies (Weikum, Moenkeberg and Hasse). This is done to achieve better performance for the database to process workloads. In order to get and keep a database system operating at its optimal performance, it must be tuned. Database tuning is done through adjusting the indexes, conceptual schema, or parameters of a database system, also known as knobs (Ramakrishnan and Gehrke). For example, if a database administrator recognizes that a certain type of query’s performance is really hurting the performance of the system, they can place an index on the table(s) to improve performance drastically.

Keeping a database running efficiently is expensive for a few key reasons. One factor that contributes to the expense of an efficient database is the database administrators who support the system. This is due to the fact that database administrators who possess the skills and knowledge required to effectively tune database systems are scarce and demand high salaries (Weikum, Moenkeberg and Hasse). In addition, database system failure and overload can cost a company thousands of dollars,
because every moment that a system is down money is lost. In order to reduce the cost of having a properly running database, a lot of commercial systems are implementing mechanisms that will do some automated tuning. This allows companies to reduce support staff for their database system, prevent some instances of system failure and overload, and reduce power consumption. Due to the complexity of some aspects of tuning there are many areas that have not been automated as of yet.

Memory contention is a classic problem that continues to draw attention from the database community. When the memory parameters have not been tuned appropriately or in the rare instances that the database’s workload is tremendously greater than the database administrators expected, database performance will be greatly hindered. The dilemma occurs in database systems when the buffer pool is not large enough to provide the currently processing queries with more than their minimum memory requirement. Thrashing, system failure, and unfairness to a certain type(s) of queries are all drawbacks.
of memory contention (Ramakrishnan and Gehrke). For example, in Figure 1 there exists a buffer pool of a database system that has a total of eighteen buffers and four processes in it. As can be seen in table 1, process one and two have a minimum memory requirement of three, while process three’s minimum memory requirement is five and process four’s minimum memory requirement is two. Because processes are greedy when it comes to memory, they all took more memory than required to speed up their processing time. When process five is added to the database system in Figure 2, it takes all of the other processes’ extra buffers as soon as it can so that it can begin processing. This causes memory contention. In this particular case, thrashing will occur, which means the system will spend more time acquiring data from disk than processing a job from in-memory data.

Much research has been done to discover ways to prevent or reduce thrashing. Finding the perfect buffer replacement policy has been the mission of many. Most of the current state of the art systems utilize some variation of the Least Recently Used (LRU)-k algorithm (Weikum, Konig and Kraiss). This buffer replacement policy strives to increase the memory hit rate. It is able to accomplish this goal by keeping the $k$ buffers with the highest probability of being accessed in shared memory at all times. The algorithm replaces the buffer with the smallest probability of being accessed when space is needed from the list of $k$ buffers. Scientists from University of Wisconsin-Madison found that controlling the multiprogramming level of each query type according to data intensity and selecting a query scheduler that works in accordance with the selected dynamic memory allocation policy for each query category will remove unfairness to different query types and minimize memory contention due to data intensive queries that
penetrate the market aggressively (Mehta and DeWitt). In order to achieve their goal, they utilized several queues to regulate the multi-programming level for each query category according to the response time of the queries. The Comfort Project utilized a feedback control loop to control the workload flow by predicting and detecting thrashing within the system based on locks held by all transactions (Weikum, Moenkeberg and Hasse). Once the system perceives memory contention, all new transactions are queued. Until the level of threat returns to a relatively safe level, queries will be allowed to enter the database gradually. When the mechanism does not perceive thrashing and allows a query to begin processing, if thrashing occurs after adding the query to the system that process will be rolled back and placed in the waiting queue.

Control theory is utilized to mathematically model the behavior of a system under various environmental conditions (Lightstone, Surendra and Diao). The system that is modeled from control theory will be stable, achieve performance objectives, and demonstrate durability if the feedback loops are designed precisely. Control theory is used in many areas because it has proved to be reliable and robust in many instances such as computer network flow control, the flight and propulsion systems of commercial airliners, as well as the cruise control mechanism in vehicles. In control theory there are two types of control loops, open and closed. The advantage of using closed versus an open control loop is that the parameters used to control the performance of the system are updated to obtain the performance objectives of a system according to its current output. The database community is beginning to believe that if control theory can be used to solve problems as complex as the ones listed above, in addition to problems not listed that are even more complex than those, then it definitely can be used in the area of
automated database tuning. We believe that control theory can be used to prevent memory contention and improve performance by controlling the multi-programming level of a database system.

Rigorous control loop design requires mathematical models that can describe system behavior reasonably well. This thesis presents the system modeling of the PostgreSQL 8.2.11 database system. This is crucial because system identification is acquired, which is one of the two steps involved in developing a closed feedback loop. Furthermore, this step provides data so that one can analyze the relationship between input and output, which allows for modeling of a system and thus, control loop design.

The rest of the thesis is organized as follows. Chapter 2 will provide a brief discussion of subject areas in which the work presented in this thesis was built upon. Chapter 3 presents the platform utilized to perform the system identification. A discussion regarding the experimental results is found in Chapter 4. Lastly, Chapter 5 presents our conclusions.
CHAPTER 2
BACKGROUND

Before presenting the model of the system, we will provide an overview of two concepts used by database systems to ensure each query is executed with minimal I/O cost. Then section 2.3 will provide examples of how non-feedback-control tuning mechanisms used to improve the performance of a system cannot prevent performance degradation from occurring for dynamic workloads efficiently in many cases. A brief discussion of the control loop options provided by control theory will conclude section 2.3 and hence the background.

2.1 Optimal Access Paths

The amount of time required to process a query in a database system is largely dependent on the quantity of data transferred to and from secondary storage systems (I/Os). This is due to the fact that computational time of a CPU is extremely fast and I/Os are slow by comparison. Therefore, if there is not a sufficient amount of main memory available to handle all the data required to process a query, the performance of the system begins to degrade (Ramakrishnan and Gehrke). This will occur because the CPU will have to wait for memory swaps to occur before it can continue executing the query. Database systems find all possible access paths using multiple algorithms\(^1\) to guarantee that it is executing every query, utilizing the access path that requires the least number of

\(^1\) This is only done when it is feasible
I/Os, as efficiently as possible. If it is not feasible to find every access path, the system uses a heuristic to find a “good enough” access path.

An access path is a plan developed from metadata provided in the database catalog on how to access memory in order to retrieve the result tuples of a query (Ramakrishnan and Gehrke). Every query has more than one access path and each one produces the same result set. In order to minimize the cost of processing a query a database selects the access path with the nominal cost. For example, assuming the states are uniformly distributed, if one desires to retrieve all tuples from the coaches’ table where state =”Texas”, which is 15% of all the tuples in the table, one access path would be a sequential scan of the entire table. However, if the table has a clustered index on the state field of the table, it would require about only 15% of the tables’ pages to be retrieved instead of 100% for the sequential search. This is possible because the index would provide the database with specific details as to what pages to read.

Join-based queries (JQs) are far more expensive and frequent than one table queries as well as all of the queries used in the experiments for this thesis. In order to construct efficient access paths for different generic scenarios, several algorithms have been developed over the last several decades to produce access paths for JQs. Below are three very popular algorithms used to construct access paths for JQs.

- Nested Loop Join the entire outer table is scanned once and then the inner table is scanned once for each tuple that exists in the outer table in order to find matches.
- Merge Sort Join each table is sorted on the match variables, then the tables are scanned in parallel to find matching tuples.
• Hash Join match variables of the outer table are turned into a hash key for each tuple and stored into a hash table, then the inner table is scanned tuple by tuple, converting its match variables into hash keys in order to identify matches in the outer table.

An access path can use any combination of algorithms in order to acquire an optimal access path for a query. The key to producing a cheap access path for join queries is the order in which tables are joined, and the algorithms used to join them must be well thought-out.

2.2 Buffer Management

As stated in the introduction, the sole purpose of database systems is to store and retrieve information for its clients. The information that the database system stores resides on external memory until it is needed. When a client sends a query to the database, the tables that correspond to the query are brought forth from external memory to the main memory (buffer pool). This is done because the CPU can access the data much quicker from main memory than external memory. The buffer manager orchestrates the process of bringing data to and from the external memory (disk I/Os) efficiently, which assists in making disk I/Os highly efficient.

In order for the buffer manager to achieve its purpose it has two key aspects, buffer maintenance system and buffer replacement policy. The buffer maintenance system is in charge of the book keeping for each buffer, knowing how many processes are currently using a buffer, and keeping track of the buffers that have been changed. Once a request is made to bring a page from external memory to main memory, the buffer replacement policy decides in which buffer to place the incoming page. The task
of the buffer replacement policy is very important and complicated, because it has to allocate the buffers in a way that will be best for all of the concurrently running queries (Ramakrishnan and Gehrke). In the past, many scientists ignored the presence of other queries processing within the system simultaneously, which lead to reduce throughput and performance because of memory contention (Mehta and DeWitt).

A database system has the ability to predict page access patterns of most queries because their page accesses are created from simple operations with very well known patterns of page accesses (Ramakrishnan and Gehrke). The buffer manager uses that knowledge to prefetch pages. For example, a client wants to view all of the records in the coach table, which is expanded over one hundred pages, and there are only fifty buffers available. First, the first fifty pages of the coaches’ table are read into the buffer pool. Once page one is read to the buffer and the CPU has processed it, the buffer can be replaced immediately by a prefetch operation of page fifty-one of the coach table. This action will be repeated until all one hundred pages have been processed, which will provide the feeling as if there were one hundred buffers available initially. If the buffer manager did not have the ability to do prefetching, the CPU would have to wait for the amount of time that it takes to execute 100 disk I/Os instead of 50. Using its ability to predict page access patterns for prefetching and buffer replacement increases disk I/O efficiency. It is also one of the key reasons why database systems do not allow the operating system to manage its memory.
2.3 Control Theory

When utilizing control theory one has to select the control loop that would be best for the problem in which a solution they desire to automate. This section will first examine several cases of open loop control database tuning methods. Then how the output performance of a system utilizing a feedback (closed) and open loop control is calculated will be shown. Lastly, an explanation as to why closed loop control is superior to open loop control for systems that have dynamic environments is discussed briefly.

Rule of Thumb Methods are very similar to control theoretical ones by their look. They are conventions implemented by people that are not guaranteed to be accurate or reliable in every situation (Weikum, Moenkeberg and Hasse). They are great for tuning knobs that impact static aspects of the database system such as the size of index pages. Knobs that require data analysis will not be able to depend on rule of thumbs. For example, in order to determine the best cache size for a workload, one would need to take into consideration the cost of the throughput of a system. There is no mathematical equation or educated guess that will provide database administrators with the solution to that question. The administrators would have to process the workload in the system multiple times and adjust the cache size to acquire the desired results.

Many database systems have mechanisms within them that improve the performance of the system by improving the settings of a specific knob. For example, the IBM DB2 Universal Database provides the Index SmartGuide that will analyze the workload and performance of the system to provide recommendations for placing indexes\(^2\) on specific tables (Schiefer and Valentin). The mechanism is great because it does in a few days what it would take database administrators weeks to determine. There

\(^2\) indexes speed up processing time by reducing I/O cost significantly
are two problems with this type of tuning mechanism. First, until the administrators give
the okay to implement the indexes, the performance of the system will not improve.
Second, when the characteristics of the workload change or increase greatly, the indexes
put in place will not help maintain optimal performance.

Some commercial systems allow companies to divide resources between user or
query categories based on some predetermined criteria. This capability affords a company
the ability to ensure fairness among all categories or give priority to the more important
categories. It also guarantees the under-utilization of resources when one category is in
need of more resources than it is allocated and another category is not. For example, a
company divides its workload into desired, valued, and important customers based on
their predetermined criteria. The resources are distributed at 50%, 35%, and 15% to the
important, valued, and desired customers respectively. Therefore, when the valued
customers are not consuming all of their resources and the important customers need
more resources at a given time, resources are wasted. There are two main causes of this
problem. The first one, obviously, is the companies provide a category with more
resources because they want them to have access to more resources in order to acquire
results as quickly as possible. Second, the workload of each category may not be static: it
can increase for a short period of time and plateau, as shown in figure 3. In fact, the
workloads of databases change constantly and are unpredictable, as shown in figure 4.
Figure 3  Workload Whose Intensity Increases as a Step Function of Time

Figure 4  Dynamic Workload
Feedback control theory provides some of the most effective tools to deal with the above dynamics in the workload. As compared to open-loop control (i.e., adaptive solutions as seen in many computer science literatures) which is frequently used in self-tuning databases, feedback loop control (closed-loop control) can adjust system status even under unpredictable environmental changes. We elaborate more on this by the following examples. Figure 5 provides a block diagram of the open-loop control while in Figure 6 a closed loop control is displayed. The reference input or desired system output, $r$, is the system workload in regards to the problem being discussed. The system output, $y$, can be the throughput or the actual or average runtime or response time. The system model is $a$. The input disturbance, $d_i$, as it refers to the problem being examined is the unexpected workload being processed by the system and the modeling error, $d_m$, is the variable response time of system output. Here the output disturbance is $d_o$. The controller determines how the input of the system should be manipulated in order to acquire the desired output, while the actuator enforces the manipulation of the input. The best perfor-

![Figure 5 Block Diagram of Open Loop Control System](image)
mance that an open-loop control could provide is $y = r \frac{1}{a} = r$ (Tu, Liu and Prabhakar).

This optimal performance for an open-loop control is contingent on an environment designed and run by Walt Disney, meaning there are no uncertainties, input disturbances, and/or output disturbances. This is due to the fact that an open loop controller does not take into account the state or performance of the system when determining its input for the system. This would be equivalent to having a lawn care provider service one’s yard every two weeks year round, even though it is only needed once a month (in most places) during the winter. But because uncertainties and/or disturbances are guaranteed to occur, the open loop system output is

$$y = r + r \frac{1}{a} d_m + (a + d_m) d_i + d_o$$

(2.1)

There are no means for the open loop controller to learn from or detect and change its input value based on the negative or positive impact that the disturbances or modeling errors have on the system, because the open loop control does not allow feedback. As one can see in Equation 2.1 the open loop system output is affected by the disturbances and modeling errors. The closed loop control utilizes controller $K$ to reduce the modeling error and disturbances introduced to the system. The system output of a closed loop is

Figure 6  Block Diagram of Closed Loop Control System
\[ y = \frac{K(a + d_m)}{1 + K(a + d_m)} r + \frac{(a + d_m)}{1 + K(a + d_m)} d_i + \frac{1}{1 + K(a + d_m)} d_o \]  

(2.2)

When \( K \gg 1 \) and \( K(a + d_m) \gg 1 \) the system output is

\[ y \approx r + \frac{1}{K} d_i + \frac{1}{K} d_o \]  

(2.3)

Therefore, closed loop controls are much better for solving problems that have dynamic environments, which have an impact on the performance of the system, because it minimizes the effects of the disturbances and modeling errors by taking into account the performance of the system.
CHAPTER 3
PLATFORM

In order for the system identification to be performed, the results of a series of experiments on that particular system have to be examined\(^3\) (Lightstone, Surendra and Diao). We embedded an actuator into the database system so that we could control the multi-programming level (MPL) (i.e. system input) and gather the output performance with great accuracy. We decided to implement this design because the ultimate goal of this research project is to use a feedback loop to control the MPL of multiple types of queries in a DBMS. Implementing the actuator in this manner required a large investment upfront, but once the system identification is acquired and a math model of the system is discovered it will be relatively easy to design the controller portion of the control loop to ensure optimal performance. Section 3.1 will briefly discuss modifications made to the original DBMS server. A very detailed description of the actuator and how it has been implemented within the system is provided in Section 3.2. This chapter will be concluded with a description of the process a query goes through to obtain results for the client and discuss modifications that were made to this part of the database system to maintain the appropriate MPL at all times.

\(^3\) See Section 4.2
3.1 Modifications of Database Management System

The Database Management System (DBMS) is the backend (brain) of the database (Ramakrishnan and Gehrke). It manages all the tasks required to start the server so that the database can be accessed. Some of those major tasks include creating shared memory and semaphores\(^4\), setting up a listening port to permit communication with the client, providing the process identification for each client, initializing data structures, etc. It retrieves the results of a query submitted to the database by a client. The DBMS is also in charge of handling system crashes when they occur by rolling back every uncommitted query, dropping all socket connections, freeing memory acquired by the backend, and restarting the server. Lastly, it orchestrates the shutting down of the DBMS so that it does not continue to consume any resources of the server that is hosting the database. The postmaster is in charge of handling most of those for the PostgreSQL database.

There were several preliminary changes that had to be made to the database server in order to integrate an actuator and perform the system identification. Figure 7 provides a block diagram of the modified system\(^5\). The actuator required more system locks and memory than the system had; therefore, we increased those parameters slightly from their default values. The initialization of the QueueManager data structure was added to the server initialization stage to facilitate the actuator. The number of client connections allowed for the system was also increased so that the system identification could be performed for a larger workload than the default setting would allow. The system shutdown process of the DBMS was updated to ensure that the message queue created for the actuator was discarded.

\(^4\) system locks utilized to protect certain variables or critical sections in parallel programming
\(^5\) Section 3.2 provides a discuss of the diagram further
Figure 7 Platform
3.2 Actuator

As stated in chapter two, an actuator is the component of the control loop that adjusts the desired parameter to the value determined by the controller. As shown in Figure 7, once the postmaster forks a child process for the client it leaves the postmaster and enters the Actuator. The actuator of this platform has two key parts. The MPL Update is the device that changes the value of the maximum MPL as directed. The second part is the Query Queue, which serves as a floodgate for the database system’s query processing engine.

The actuator is set to update the maximum MPL after a relevant period time $t$, which is determined by the database administrator. The maximum MPL value represents the maximum number of queries that the database can process simultaneously. In order to keep track of the time, a time variable is set at the startup of the server. Every time a query enters the queue or finishes processing, the actuator checks to see if $t$ time has passed. The semaphore created to protect the maximum MPL is exclusively locked so that no other process will try to update it. When it is time to update the maximum MPL, then the maximum MPL is set to a predetermined value. When the MPL is increased, a message is sent to the first item of the Query Queue. But if it is decreased, no more queries will be allowed to exit the queue to run until the current MPL is less than maximum MPL. Then the process proceeds to see if the system will allow the query that just entered the system to exit the actuator and run. If it cannot, it will be passed directly to the Query Queue.

The Query Queue consists of all queries waiting to be processed. When a query is received by the DBMS, if it is not time to update the MPL and the current MPL is greater
than or equal to the maximum MPL, it is placed into the Query Queue as presented in
Figure 7. Because processes run in parallel and we desired to add only one query to the
queue at a time, a semaphore was created to prevent multiple queries from being added
and or removed from the queue simultaneously. We utilize a light weight semaphore
because we did not want the query waiting on the lock to consume resources looping
until it was its turn to be added or removed from the queue. Queries queued are granted
access to run on a first in first out (fifo) basis. In order to prevent wasting resources by
creating a while loop that would loop until the current MPL was less than the maximum
MPL, another light weight semaphore was created to prevent more than one query from
trying to update the current MPL at the same time. There are two challenges that had to
be taken into consideration when trying to devise a plan as to how to alert the first item in
the Query Queue that it was time for it to run. The first challenge is the while loop would
loop until the current MPL was less than the maximum MPL, meaning resources are
being wasted on looping for as much as five minutes. The second challenge is that
because this is a child process, when the maximum MPL is updated to a greater value or
current MPL has reduced to a value less than maximum MPL, it does not find out about it
immediately unless there is correspondence with the parent process within the loop. For
example, a print to screen statement would have to be executed for each iteration of the
loop. Therefore, we use a system message queue that allows the process to remain
sleeping until it receives a message acknowledging that the current MPL is less than
maximum MPL. Then the head query of the queue response time is calculated and
allowed to run. Once the database has processed the query and it is ready to send the
results to the client, the runtime of the query is calculated and the current MPL is
decreased. If the current MPL is less than the max MPL and there is a query in the Query Queue, the process sends the "GREEN LIGHT" message to the first item in the Query Queue. Then the query exits the actuator to be processed.
CHAPTER 4
EXPERIMENTAL RESULTS

This section will provide a discussion of the experiments and results. First the workload will be presented. Next the details of the experiments executed will be discussed. Lastly, an interpretation of the experimental results will be provided.

4.1 Workload

The TPC-H benchmark was used as the workload for the experiments. In order to simply verify that our platform is valid, we will utilize all of the queries whose minimum memory requirement was less than 100 MB. Table 3 provides a list of the 16 out of 22 queries that were used in our experiments.

<table>
<thead>
<tr>
<th>Table 3 TPC-H Benchmark Queries Utilized</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,3,4,6,7,8,9,10,12,13,14,15,16,18,19,22</td>
</tr>
</tbody>
</table>

4.2 System Identification

System identification is one of two crucial stages required to develop a system model that is based on a closed control loop (Lightstone, Surendra and Diao). The sole purpose of this stage is to define the relationship between the input and output of the system based on the values of measured output as responses to the manipulated variables (Franklin, Powell and Workman). When done correctly, the system identification stage produces the system model required for the controller design. System identification can
be acquired through two avenues: one could either run experiments for every possible value of every parameter or create an equation\textsuperscript{6} based on data collected from a few experiments (Lightstone, Surendra and Diao).

In order to collect the data required to create the equation for our database, our experiments consist of a series of step function tests. Step function tests expose how the system reacts to a sudden change of the input signal. Specifically, we set the multiprogramming level for a database workload and change it significantly after a predefined period of time. There were a total of 6 different experiments run. The system was fed a certain number of queries consistently in order to keep some queries in the waiting queue. This allowed the performance of the system to slow down or speed up, without the system having too many connections or not enough queries to meet the current multiprogramming level. Table 4 presents the test number, initial multiprogramming level, and final multiprogramming level (MPL) of the six step function tests that were executed.

<table>
<thead>
<tr>
<th>Test#</th>
<th>Initial MPL</th>
<th>Final MPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>Test 2</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Test 3</td>
<td>10</td>
<td>25</td>
</tr>
<tr>
<td>Test 4</td>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>Test 5</td>
<td>25</td>
<td>50</td>
</tr>
<tr>
<td>Test 6</td>
<td>25</td>
<td>80</td>
</tr>
</tbody>
</table>

\textsuperscript{6} Statistical model
4.3 Results

After examining the results of tests 1-6, which are graphically displayed in figures 8-14, we found that test 1 was the only experiment where the system reached a steady state. The steady state of a system is simply the time in which the system’s performance levels out. These results fostered two conclusions. First, the graph of the results in figure 10 has a positive slope, which facilitates our conclusion that the system can be described by a second order model. The equation chosen to decipher the relationship between the input and output based on those results was equation 4.1, the

\[ G(s) = \frac{a}{s^2 + bs + c} \]  \hspace{1cm} (4.1)

transfer function, because it is a standard function used to describe second order model systems. The transfer function has three parameters \( a, b, \) and \( c. \) Second, the lack of system stability shown in tests 2-6 makes us believe that the maximum MPL that our system is able to process between 15 and 20, if a reasonable response time is desired. Therefore, we utilized the results of test 1 to calculate the parameters of the transfer function for our system. This was done because if one chooses an experiment that does not reach a steady state, then the controller produced from the transfer function will never select a system input that will provide the desired performance results.

\[ G(s) = \frac{kwn^2}{s^2 + 2zwns + wn^2} \]  \hspace{1cm} (4.2)

Utilizing standard control analysis techniques, the transfer function\(^7\) becomes equation 4.2 for our system, where it is clear that \( a, b, \) and \( c \) have been replaced by \( wn^2, 2zwn, \) and \( wn^2 \) respectively. The dc gain is represented by \( k \) and provides insight as to

---

\(^7\) Equation 4.1
how the system reacts to a sudden input increase. We were able to calculate the dc gain by dividing the steady state value of the response time by five. A damping factor shows the system’s ability to minimize the fluctuations in achieving stability. In order to take into account how well the damping mechanism of a system performs after an increase is made to the input, the damping ratio, $\zeta$, is also utilized to provide a sound transfer function. The value of the damping ratio was easy to calculate because we were able to calculate the overshoot of the value using the results of the experiment to find the peak response time value as well as the steady state value. The natural frequency, $\omega_n$, of the system affects every parameter because it is a direct reflection of its stability and performance. After analyzing the results to find the frequency of the response time, the natural frequency was calculated using the frequency and the damping ratio. Hence, the model of our system in regards to the response time is

$$G(s) = \frac{0.029152}{s^2 + 0.007304s + 0.000146} \quad (4.3)$$

The process that was done to find the model of our system in regards to the response time was also done to find the model of the system in regards to the runtime by using the runtime results of test 1. Therefore, we were able to determine that the model of our system in regards to the runtime is

$$G(s) = \frac{0.035367}{s^2 + 0.006015s + 0.000141} \quad (4.4)$$

---

8 The value of the mpl increase
Figure 8  Test 1 MPL Goes from 10 to 15

Figure 9  Test 1 Second Order Model

Figure 10  Test 2 MPL Goes from 10 to 20
Figure 11  Test 3 MPL Goes from 10 to 25

Figure 12  Test 4 MPL Goes from 10 to 30

Figure 13  Test 5 MPL Goes from 25 to 50
Figure 14 Test 6 MPL Goes from 25 to 80
CHAPTER 5

CONCLUSIONS

Automated database tuning is the key to driving down the cost of maintaining database systems for corporations. We have argued that a feedback loop control, where applicable, is the best concept one can use to implement automated tuning because it guarantees stability, desired performance, and robustness. In this thesis, we present a platform that acquires the system identification and defines the stable MPL limit of a system, by modifying the database management system of PostgreSQL. It is our goal that as a result of providing a platform that acquires the system identification, which can build the foundation for rigorous controller design, more scientists will begin developing automated database tuning solutions using control theory, specifically feedback loop control.
REFERENCES


