A Performance Analysis Framework for Database Management Systems

José Antonio Fernandes de Macêdo
e-mail: jmacedo@inf.puc-rio.br

Philippe Picouet
École Nationale Supérieure des Télécommunications de Bretagne (ENST-Bretagne)
e-mail: Philippe.Picouet@enst-bretagne.fr

PUC-RioInf.MCC40/04 November, 2004

Abstract: Performance evaluation of DBMS is a major issue since it is generally difficult to model experimental and performance analysis results. In this paper, we propose an application framework that provides a model, a methodology and a common platform to implement database evaluation analysis tools. Inspired on a conceptual UML model [14], this application framework provides a much more detailed model that allows capturing the complex structure of DBMS modern software. We use a recent work [1] about the implementation of a new data page layout to illustrate the instantiation of our framework.

Keywords: database management system, performance evaluation, performance analysis, application framework.

Resumo: A avaliação de performance de SGBDs é uma importante questão a ser tratada dada à dificuldade de modelar o que deve ser medido e analisar os resultados obtidos. Propomos neste artigo um framework de aplicação o qual fornece um modelo, uma metodologia e uma plataforma comum para a construção de ferramentas análise de performance de banco de dados. Inspirada em um modelo conceitual descrito em UML [14], este framework provê um modelo que permite capturar a estrutura complexa de um SGBD. Usamos um recente trabalho [1] sobre implementação de formatos de página de dados para ilustrar a instanciação do nosso framework.

Palavras-chave: sistema de gerencia de banco de dados, avaliação de performance, análise de performance, framework de aplicação.

SUMÁRIO

1. Introduction

Quantitative performance evaluation is a major concern of computer science research and systems, especially for complex architectural software such as DBMS [13, 7]. Since new algorithm and platform evolve regularly, it is necessary to compare them to older technologies in evaluating the such difficulties, we can find both the evolution of the environment and the complexity of the software architectures: the evolution of the environment covers, for example, the hardware progress, which makes it almost impossible to regenerate a previous experimental context; the complexity of software architecture denotes the difficulty to identify the impact of software service implementations with complex interactions in a performance evaluation.

Although classic performance evaluation models are well known, there is neither a unique methodology nor a tool that supports the performance analysis tasks. The result is twofold: on one hand, the research papers proposing some specific improvement techniques [9, 6, 15] describe some performance analysis comparisons which are difficult both to produce and reproduce; on the other hand, commercial DBMS have many difficulties to provide software adapted to specific platforms [12].

In this paper, we focus on these difficulties and propose a framework to support design, execution and reuse of performance evaluation models. Like [16], we believe that such performance analysis study should be conducted from the very beginning of a certain product development and we believe that such framework could help to reach this step.

In Section 2, we analyze the difficulties involved in the production of a quantitative performance study over complex software architecture and especially over DBMS. We describe the main architecture of our generic framework and explain how to specialize it for DBMS in Section 3. In Section 4, we describe the methodology associated with our framework by applying it to a recent study [1] proposing a new data page layout. We finally conclude in Section 5.

2. Performance Analysis of DBMS

Recent publications in DBMS performance analysis [1, 2, 3, 5, 6, 9, 11, 13, 15] have shown that the formalization of a performance model is essential to guarantee a good interpretation of the results. Although the performance model plays a major role in database research, there is not any generic performance model to evaluate DBMS behavior. In general, when it is necessary to elaborate a performance model of any system we must answer the following questions:

What software component[1] to measure? The software component that must be measured can be of different granularity: the entire system (i.e. DBMS), a component implementation, a data structure or a specific algorithm. For example, in [13], transaction processing and database benchmark permitting measurement and comparison between different commercial DBMS performance are proposed, while in [15] we propose techniques for buffer accesses to memory-resident tree-structure indexes, where the efficiency of the different B+-trees is measured in order to avoid trash cashing.

What experimental platform to choose? In order to execute the performance analysis we have two possibilities in choosing the experimental platform: simulation artifact or real system. The simulation artifact simplifies the execution of the performance because it reduces the number of components we must deal with regarding the real system. However, it is hard to ensure that the simulation model is credible, if the simulation is accepted as being accurate and useful. On the other hand, the use of a real system is the most reliable and preferred way to validate the performance analysis but it is more complex to build, analyze and reuse. Generally, publications utilize their own DBMS to execute their experiments because it gives them more control during the test [11, 1, 5]. In absence of DBMS, a simulating environment is used in some publications [6].

What measures to collect? Although some publications in the past defended that it is sufficient to use simple measurements such as elapsed time, CPU time and I/O activity [18] to measure performance, in actuality we see that the influence of new platform features in the DBMS

[1] A software component is a software technology for encapsulating software functionality [17]

performance is crucial [5,6,2]. Consequently, we must take into account the operating system, processor and device variables to model the DBMS behavior.

How to interpret data results? As the performance analysis expands outside the DBMS, it needs help on how to effectively use data results to enhance the identification of performance bottlenecks. Most importantly, the performance analyst must be trained on how to interpret performance data so that the limitations of the measurement process can be clearly understood. Thus, cost models are required to characterize the performance metrics. Generally, the performance metrics use probability [6], queuing networks [19] or equation systems [5] models.

In addition to the challenges described above, the DBMS also has some particularities that make the performance modeling more complex:

The DBMS is situated between application requests and platform services (operating system, network, devices, etc) that force the DBMS to deal with the complexity of this environment. Besides, the heterogeneity/complexity of the applications and platforms makes the description of performance properties via small sets of metrics difficult;

The multi-layering and variety of DBMS software architectures and the diversity of software components and algorithms found in a DBMS makes it hard to construct simple and portable benchmarks. Also, it is difficult to identify and measure the performance of a specific component inside a complex software architecture;

The flexibility of DBMS configuration makes it difficult to provide a concise representation of system resources. The large set of system parameters makes the system modeling and analysis difficult.

As previously mentioned, although there is a lot of work that makes use of quantitative performance analysis in the DBMS area, we identified a lack of methodology and common platform to build a performance analysis model. This fact forces the construction of a performance model from scratch each time. In summary, we need a methodology to guide the construction of a performance model and a system that permits its implementation and execution. Thus, the Object Management Group (OMG) has defined a UML profile [14] that enables the design of performance models. This model is based on the identification of workload, resources and scenarios. However, this model does not present all service implementations needed to instantiate an application for several reasons:

First, in complex environments such as DBMSs we need to analyze facts through different logical views independently of the software physical implementation. Thus, we suggest the conversion of the UML resource concept into service and service implementation concepts. The service represents a logical view of the implementation artifacts named service implementation;

Second, in multi-layered architectures we need hierarchical mechanisms to represent the relationship complexity. The new model must allow the description of software layers and their complexity;

third, it is necessary to measure the system service implementations quantitatively using a metrics model; It is necessary to have in the model classes that allow the definition of complex formulas to represent the wide range of cost models available;

Fourth, we need a performance model to implement, execute and reuse. In this manner, the model must provide mechanisms that permit the execution and evaluation of performance models, such as measurers and execution engines.

We claim that an application framework can fit all these requirements [8]. Based on the OMG UML Profile model [20], we transform this model into an application framework able to support the creation of performance analysis tools geared towards the DBMS analysis. We identify the following five elements fundamental to performance models: scenario, workload, service, service implementation and metrics. Figure 1 illustrates the framework overview: a scenario drives a workload execution; a workload represents a unit of job with applied load intensity that runs over a service; a service is the logical view of software physical implementation; all these layers can use some kind of metrics to model their behavior.

3. A Generic Framework for Performance Evaluation

In this Section, we describe the framework sketched in the previous section. Based on the five previously identified elements (scenario, workload, service, service implementation and metrics), our framework UML packages (Figure 2) recall that, besides the metrics package, the scenario package drives all the other packages, but relates to them through different dependencies. Two kinds of dependencies are stressed to differentiate the design and the execution of the performance framework: from a conceptual point of view, performance models are represented as layered hierarchies relating application scenarios (at the top) to software implementation (at the bottom) through specified workloads and service specification;

From an execution point of view, all these layers can use some metrics to model their behavior. The scenario is restricted to a set of workloads, services and service implementations that will be executed in order to measure the system in a defined condition.

We will now detail each package showing internal classes as *Scenario*, *Workload*, *Service* and *ServiceImplementation* (Figure 3). These classes are hierarchically modeled using the design pattern Interpreter/Composite [10] that enables the implementation of hierarchical structures with the expressive power of a regular grammar. Elementary classes are the leaf classes of these hierarchies, which are built thanks to group classes: the relationship between a group class and its superclasses, for example *GroupScenario* class and *Scenario* class, allow defining an ordered sequence of elementary action.

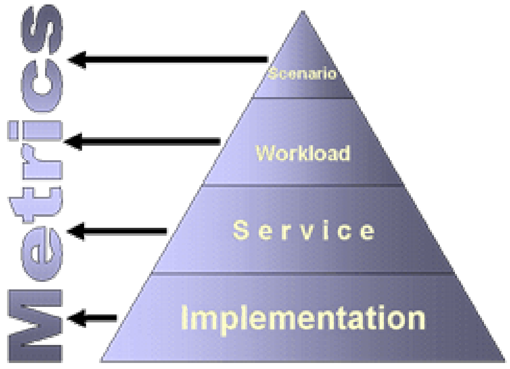
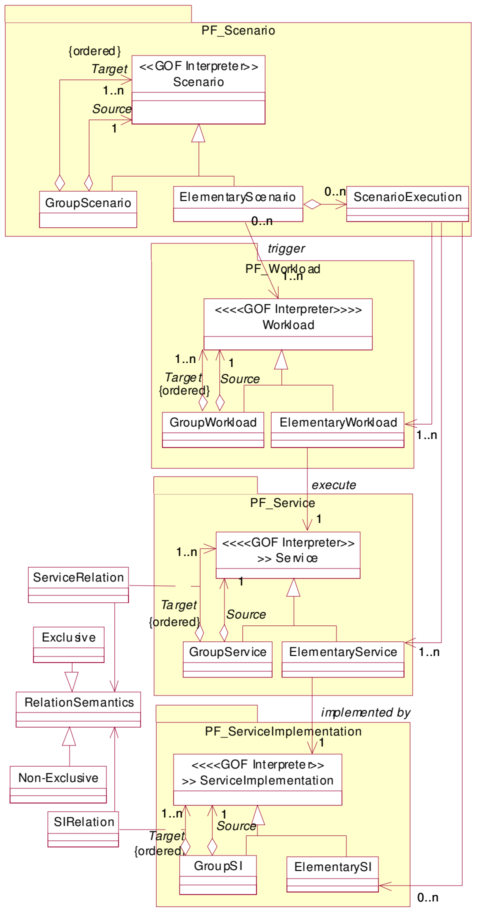


Figure 1 - Framework Overview



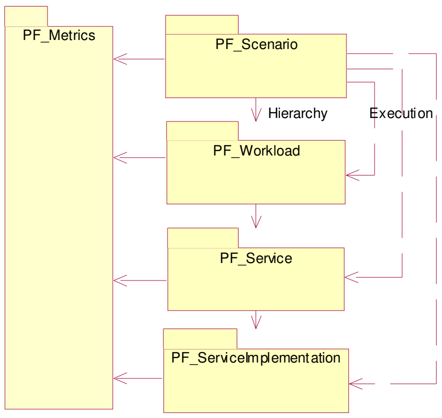


Figure 2 - Framework Packages

Figure 3 - Framework Packages Relationships

An elementary scenario is composed of a set of *Workload*, *Service* and *Service Implementation* class instances that delineate the specific context that must be evaluated. The *Workload* class aims at defining a set of operations to be executed over the system. Single operations are denoted by the *ElementaryWorkload* class, and intuitively connect to corresponding services. The *ElementaryService* class is dedicated to modeling a specific service carried out by the system. Services allow defining a logical view of service implementations independently of the physical implementation and organization aspects. With this approach, it is possible to model system functionality in a great variety of ways. Also, the service represents the dynamic portion of the system while the service implementation definitions focus on its static structural portion. A service implementation can define concrete system components such as processors and devices (i.e., cache, memory, disk, etc) or abstract ones such as database modules (i.e. buffer manager, transaction processor, etc). The *ElementaryService* class is associated with a software component that implements it, represented by the *ServiceImplementation* class.

Concerning the *Service* and *ServiceImplementation* classes, both have associative classes that represent the semantics (exclusive or non-exclusive) of this relationship. For example, a query service can be related with two types of optimization that are exclusive (conceptual point of view). Thus, during the execution of a scenario (*ScenarioExecution* class), only one type of optimization will be executed.

3.1. Performance Metrics

The framework provides some classes to assign formulas to each element model (workload, service and service implementation) as we can see in Figure 4 (there is not any formula for the scenario since they only compile results from other elements).

In order to allow the definition of regular expressions, we have defined the *Formula* class that follows the Interpreter design pattern [10] (Figure 4).The *Formula* class may contain an expression or a variable that delineates the quantification of an element. This class permits us to define group expressions (*Expression* class) and elementary variables (*Variable* class). A group expression can be a unary, n-ary and boolean expression. These classes are also framework hotspots that can be extended to new types of expressions depending on the type of systems (see below). For example, a workload instance may define an execution timeout variable that is equivalent to two seconds; the query service may measure the query execution stall time that is the sum of the processor, cache and memory stall time. Figure 4 illustrates the relationships between *Workload*, *Service* and *ServiceImplementation* classes with *Formula* class. We can observe that the *Workload* class must declare one or more formulas and the *ServiceImplementation* class may have no formula associated. Note that the *Service* class has two links to the *Formula* class that symbolize the variables that are received as input and the variables that are sent as output, permitting the definition of the service interface and the transformation expressions. For example, the query service receives the number of queries to be executed and calculates the amount of execution time and number of table tuples returned.

In addition, the variables can make use of predefined units and types symbolized by *Unit* and *ValueType* classes. Thus, it is possible to declare new attributes to the workloads, services and service implementations elements.

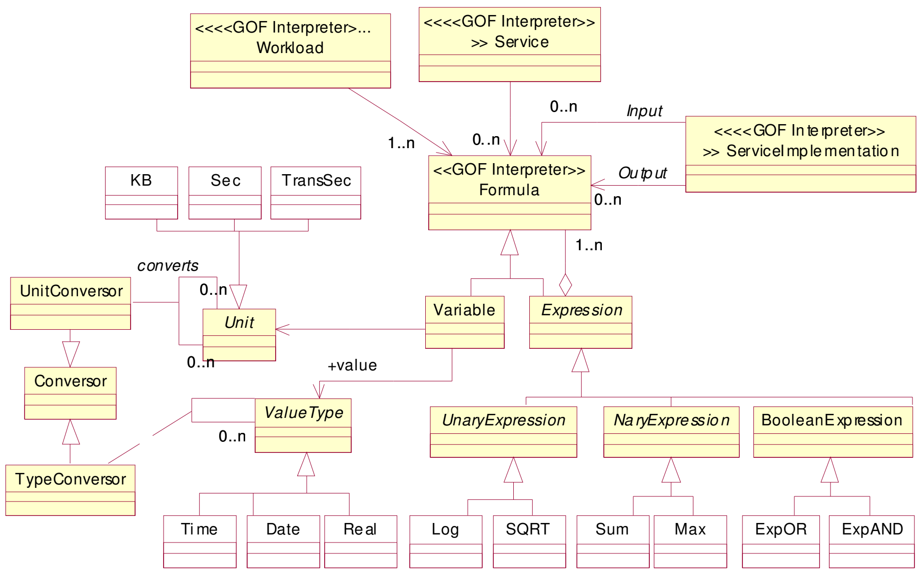


Figure 4 - Cost Expression

3.2. Evaluating Performance Model

Finally, the ability of the framework to validate the performance model is obtained by executing the scenarios, collecting the results and making scenario comparisons. The framework proposes partially implemented methods that must be used to compute the model evaluation.

Schematically, the *evaluate* method of *PerformanceContext* class starts the evaluation. It calls the *evaluate* methods from scenario objects that are propagated to the evaluate method of *Workload*, *Service* and *ServiceImplementation* classes. An evaluate method of a specific service implementation can contain custom code to call external software such as DBMS or an OS program because some evaluations may need to use real systems to collect measures into their formulas. Likewise, the evaluate method can contain simulation code to generate data to be used by cost formulas. After each *evaluate* method execution, it is necessary to treat data and compute the cost formulas. The data manipulation and formula computation are made by the *collectCost* method presented in the *Scenario*, *Workload*, *Service* and *ServiceImplementation* classes. During execution, the result of each method is propagated to the caller method until it reaches the *PerformanceContext* class where it is shown. Each evaluation method may include custom codification to make coherent interpretation of variables and formulas.

Scenario comparison is an important features provided by our framework. A scenario might be seen as an execution plan that can be measured in several points. The *ElementaryScenario* class (Figure 5), which represents a specific service, can associate diverse measurers that will collect corresponding measures during the evaluation execution. Each measurer attaches some start and stop measurement points associated with a service (or service implementation). Also, a measurer is defined as an expression that must be evaluated. For example, we can have a measurer that collects the elapsed time from the start of query execution until the first page stored in the memory.

The *Comparison* class associates a source scenario that must be evaluated and compared to diverse target scenarios. The possible types of comparisons are represented by: *Straight*, *Contrast* and *Analogy* classes. The straight comparison denotes that we want to emphasize both similarities and differences while the contrast comparison emphasizes mostly differences and the analogy mostly similarities.

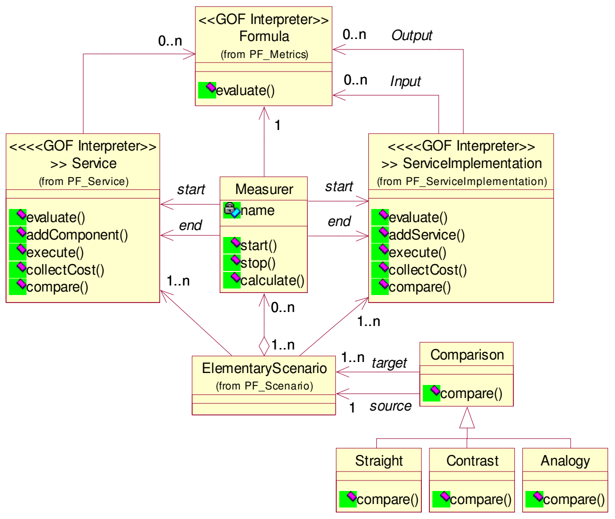


Figure 5 - Scenario Comparison

3.3. Adapting the framework to DBMS

In this section, we propose some specialization of the former framework to Database performance evaluation. More accurately, we will describe some specialization of the previously described *Workload*, *Service* and *ServiceImplementation* classes. These specializations will be used in the next section to describe the implementation of a performance evaluation.

Adapting the *ElementaryWorkload* Class

In a DBMS context, the *ElementaryWorkload* class may be specialized in *DBMSWorkload* class that represents a typical workload of the DBMS as illustrated in Figure 7. A DBMS workload is composed of a transaction (*TransactionType* class) that uses a data set (*DataSet* class). The *ElementaryWorkload*, *TransactionType* and *DataSet* classes are examples of framework hot spots that may be extended according to the application needs. Figure 7 illustrates some possible extensions of theses classes such as the *TPCDataSet* class that characterizes the TPC Benchmarks data sets [13] and the ZipfDistribution class that may be used to generate a data set using zipf distribution [20].

Adapting the *ElementaryService* class

As we have observed before, services allow defining a logical view of the service implementations independently of the physical and organizational aspects. There are many works concerning performance analysis of services either at large granularity (like query processing, buffer usage [1,2,3,5,6,11,15]) or at low granularity services (as algorithms [9]). Thus, we have extended the *ElementaryService* class with the *DBMSService* class (Figure 6) including some typical DBMS services. This class can be extended with another kind of services.

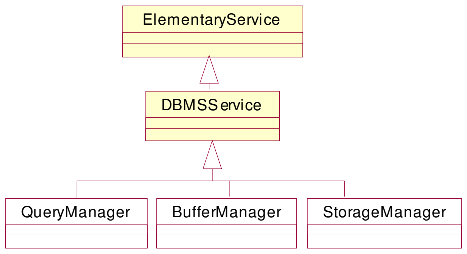


Figure 6 - Service Hierarchy

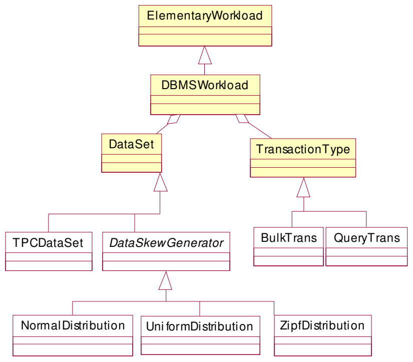


Figure 7 - Workload Hierarchy

Adapting the *ElementarySI* class

In our framework, we have divided the *ElementarySI* class as platform components, DBMS components and algorithms (see Figure 8), that respectively correspond to the *PlatformComponent, DBMSComponent* and *Algorithm* classes. The *PlatformComponent* class represents the external components from the DBMS in contrast to the *DBMSComponent* class that describes the internal DBMS components. We have specialized in the *SOComponent* class representing the memory hierarchy. The *DBMSComponent* class specialization will depend on the DBMS components we want to investigate in the performance evaluation.

The *Algorithm* class has been specialized for DBMS algorithms purposes. An algorithm is represented by the *DBMSAlgorithm* class that is composed of algorithm components. We have defined the algorithm component as a sequence of operations (read, write, etc), data structures (list, B-tree, etc) and reference patterns. The reference pattern clarifies how a set of algorithm operations manipulates a specific structure because it influences the algorithm performance. Also, we have expressed the data distribution and data volume through the *DataSkew* and *DataVolume* attributes because the algorithm performs changes according to these two factors, for example some of the currently most useful algorithms (e.g., sample sort, block radix) are dependent on the data distribution [4].

The *PlatformComponent-Algorithm* division aims at describing the service implementations according to different grades of granularities. Besides, the division of the algorithm in sub-parts permits us to compare the structure of several algorithms at different levels of granularity. The granularity level that must be used on the framework depends on the performance analysis' goals: for example, if we analyze the impact of a new buffer replacement algorithm, we have to define the algorithm service implementations and compare different implementations. However, if we want to identify some service performances or possible bottlenecks, we only need to describe services. An important aspect of our framework is that we can progressively refine it using different levels of granularity to obtain more details of DBMS behavior.

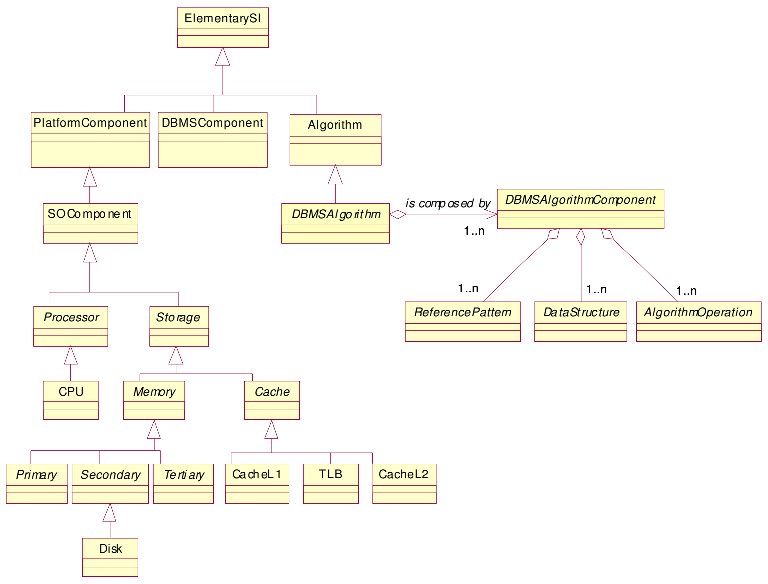


Figure 8 - Service implementations

4. Cache Performance Analysis – DeWitt Model

To describe the use of our framework, we will apply it to [1], a performance analysis over cache operation, and show how to instantiate our framework to implement this evaluation.

In [1], the authors study the impact of a new type of data page layout (PAX, which fits better the memory cache) on query execution time. The proposal is validated by a comparison of three data page layouts (PAX, NSM [22] and DSM [21]). Measurements have been done on a specific storage manager and demand the implementation of the three page layouts.

The experiments aimed at validating data manipulation algorithms such as insert, update, delete and two query operators: scan and join. Experiments were conducted on a specific platform with identified characteristics. The workload consisted of one relation (TPC-H fashion) and variations of the range selection query. The methodology to collect the experimental measures was made by reading two counters provided by one processor tool.

We show in that Section that our application framework can help build the performance model. Also, we aim at demonstrating the advantage of following an implicit methodology derived from the framework instantiation process. Consequently, the designer has to follow a four step process to instantiate the framework:

1. Extend the framework hot spots;
2. Describe the DBMS algorithms;
3. Create scenarios;
4. Define the cost formulas.

Below we describe each step showing how to derive the framework into a concrete architecture. Due to space limitations, we do not provide exhaustive details and restrict the presentation to the specific scenario of range selection queries on a memory-resident relation.

4.1. Step 1 – Extending the framework hotspots

In this first step, we must re-examine and concretize the hotspots defined in the framework, which are: workloads, services and service implementations. Generally, a hotspot is made of abstract classes that must be specialized in concrete classes [8]. However, there are some hotspots that are made of concrete classes, so that we can specialize these classes or use them as they are.

The implementation used in DeWitt's work is made specifically using an appropriate storage manager (SHORE). Thus, there is no need to define other DBMS services (This peculiarity shows another advantage of the framework: the possibility of modeling only the important services we want to evaluate). This approach is fundamental in order to generate a performance model that is not biased from specific proper implementations. Consequently, we define:

the storage management service as shown in Figure 9(a) and extend it to represent specifically the SHORE storage service

a class named *SHOREStorageComponent* that symbolizes the service implementation responsible for receiving a query and execute it in the SHORE storage manager (Figure 9(b)).

the workload (*PAXWorkload* class, Figure 9(c)) that will trigger this service is composed of a range of select queries executing (*RangeSelectQuery* class) over a table created specially for the test, which we call *PAX_Table*.

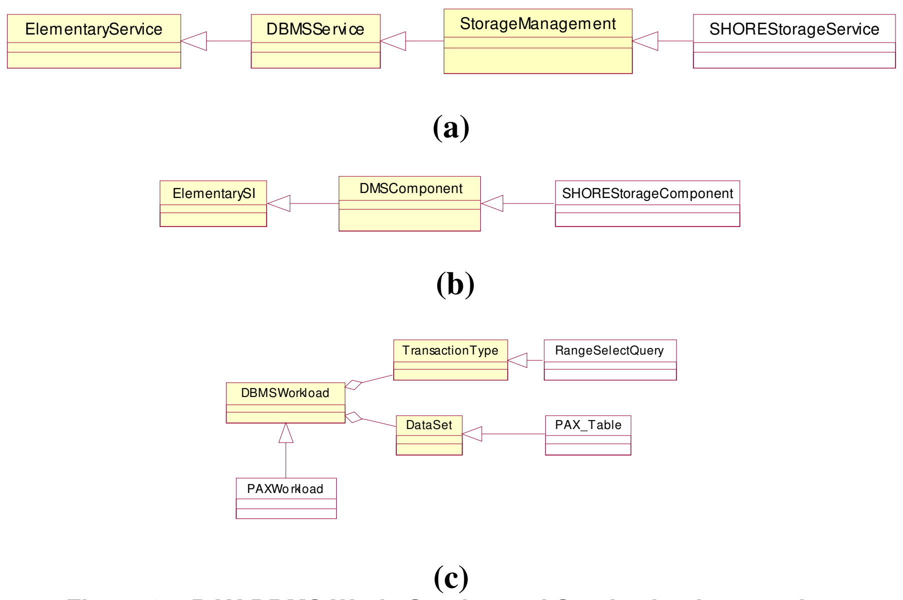


Figure 9 – PAX DBMS Work, Service and Service Implementation

We must also declare the platform components that are necessary for the performance context such as cache, CPU and memory. For example, the type of machine (Dell 6400 PII Xeon/MT), the cache features and the main memory capacity could be modeled by a class specialization shown in Figure 8.

4.2. Step 2 – Defining the DBMS algorithms

This step is only mandatory if you investigate a specific pre-determined algorithm. In such case, we have to specify each algorithm detailing all internal service implementations such as operations, data structures and reference pattern.

Concerning DeWitt's work, the relevant DBMS algorithm concerns the storage manager service related to the task of reading and writing data pages. Then, we extend the *DBMSAlgorithm* class and define the *AlgorithmDataStorage* subclass (Figure 10). For each type of data page layout (PAX, NSM and DSM) we create a new specialized class and define its data structures and its operations (read and write).

4.3. Step 3 - Creating Scenarios

The definition of scenarios can be made at design time or execution time, respectively by the performance designer during the instantiation of framework, or the user of the performance tool during the execution of the evaluation performance. These approaches depend on the type of performance tool we want to build. For example, the definition of the scenario during the execution needs the construction of a complex user interface that must manipulate diverse hierarchies.

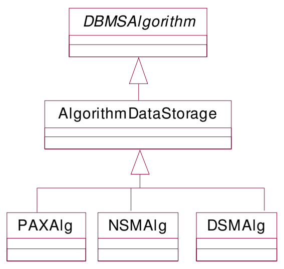


Figure 10 - PAX Algorithms

```
// Method of PAXScenario class
void configure ( ) {
        PAXWorkload pw = new PAXWorkload(PAX_Table , RangeSelectTable);
        SHOREStorageService sss = new SHOREStorageService();
        sss.addComponent(new SHOREStorageComponent());
        NSMAlg nsm = new NSMAlg();
        DSMALg dsm = new DSMAlg();
        PAXAlg pax = new PAXAlg();
        sss.addComponent(new GroupSI(nsm,dsm,pax, new exclusive( ));

}
```

Figure 11 - Configure PAX Scenario

For each scenario that we want to evaluate, we have to describe a concrete class that extends the *Scenario* abstract class. Each extended class will implement the configuration of the specific scenario describing workload, service implementation and service configuration (Figure 11 - Configure PAX Scenario). Then, we implement the configure method of each scenario. This implementation describes a particular configuration through the instantiation of *Workload*, *Service* and ServiceImplementation classes already defined in the first step.

In our example, we want to create the *DWScenario* class, which makes a comparison between PAX, DSM and NSM. In this case, Service is limited to the storage service and modeled as a subclass of *DBMSService* class (called *SHOREStorageService*). The corresponding service implementation is modeled as the *SHOREStorageManager* class (Figure 9 – PAX DBMS Work, Service and Service Implementation(b)) that is a subclass of the *StorageManager* class (an elementary service implementation). The workload is modeled by the *SHOREWorkload* class.

Then it is necessary to associate each workload with the service that it triggers and, for each service, the corresponding service implementation. The code fragment bellow (Figure 11 - Configure PAX Scenario) illustrates the implementation of the configure method of the *DWScenario* class.

4.4. Step 4 – Defining metrics models

The metrics model is essential for the framework instantiation because it defines the cost formulas associated with workload, service and service implementation elements. These formulas are used by the system to evaluate the performance, permitting the comparison between different executions of scenarios. As presented in Figure 4, formulas are regular expressions and we can extend the Expression class to represent a wide variety of functions that are needed to create the performance model. In DeWitt's work, the formulas presented are simple sum expressions that calculate the execute query time. The formula used is: Tq = Tc + Tm + Tb + Tr – Tovl, where Tq is the time to execute a query.

As shown in

Figure 12, the variables that will be used by composing the cost formulas are created as extensions of the Variable class. Their values are collected from the processor execution via a proprietary program. We show below how to implement the external program call. Moreover, we capture program variables to fill up the cost formulas. Figure 13 shows how to implement the formula in the configuring method of the Scenario class. We can see that the *CacheMiss* and *StallTime* variables are used by *Mult* and *Sum* classes to create the expression objects.

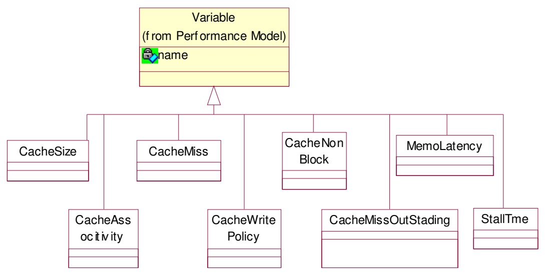


Figure 12 – Variable Especialization

```
void configure ( ) {
   …
   Expression TL1D = new Mult (CacheMiss, 4);
   Expression TL1I = StallTime;
   …
   Expression TM = new Sum(TL1D,TL1I,…);
   …
   Expression TQ = new Sum(TC,TM,TB,…);
}
```

Figure 13 - Formula Codification

An interesting characteristic of DeWitt's work is that the measurement of system behaviors is made outside of DBMS by an appropriate program, even though the implementations were made into the DBMS. This approach is consistently justified since the objective is to study the cache performance, which is managed by the computer processor. In such case, the best cache information is naturally collected directly from the processor via a specific program. This is achieved in our framework by the implementation of the *evaluate* method. In the case of DeWitt, it is done by programming the access to the external resources in the *evaluate* method of *SHOREStorageManager* (Figure 9) and *CPU* classes (Figure 8). Figure 14 is shows the *evaluate* method of the *SHOREStorageManager* class that connects with SHORE and executes a query. Figure 15 shows the call to the external program to collect the processor measurements and the variable assignment.

```
// SHORE Storage Manager class
void  evaluate ( … ) {
    …
   connect( login, pass);
   Execute_Query (tx);
   Disconnect( );
   …
}
```

```
// CPU class
void  evaluate ( … ) {
    …
   Execute_Program ( );
   data = CollectData( );
Scenario.setVariable(data);
    …
}
```

Figure 14 - Executing Transaction on SHORE

Figure 15 - Executing Processor Program

5. Conclusions

In this article, we proposed generic application framework architecture to achieve a quantitative performance evaluation on DBMS.

This framework is based on five basic abstractions of performance analysis: scenarios, workload, service, service implementation and metrics. The relationships between these abstractions fit well the representation of a complex multi-layered architecture. Some extensions of the model for DBMS have also been proposed.

Thus, our framework is associated with a methodology that guides the designer of a performance evaluation study to achieve his/her goals and to produce a concrete evaluation platform. We have demonstrated this final characteristic on a recent data page layout study [1]. We have shown that our framework also permits making comparison evaluations between alternative service implementations.

We believe that such a framework should provide some interesting benefits to achieving the following goals:

- capturing performance requirements within the design of performance context;
- executing external applications in order to collect measures and integrate the performance analysis with disconnected service implementations;
- Specifying metric models for the workloads, services and service implementations.

Further studies have to be done. As we have noted in Section 2, our work is partly inspired on UML profiles and a deeper study on how that model and our framework could complement each other would be interesting. Moreover, since the UML profile is dedicated to modeling any kind of software, it would be interesting to evaluate how generic our framework is and to study if it can be specialized for other complex software domains.

References

1. Ailamaki, A.; DeWitt, D.; Hill, M; Skounakis, M.; Weaving Relations for Cache Performance; The {VLDB} Journal, p.169-180; 2001.
2. Ailamaki, A.; DeWitt, D.; Hill, M.; Wood, D.;DBMS on modern processors: Where does time go ?; International Conference on Very Large Databases (VLDB); 1999.
3. Boral, B.; DeWitt; D.; A Methodology for Database System Performance Evaluation; Technical Report; University of Wiscosin; 1983
4. G. Blelloch et al. A Comparison of Sorting Algorithms for the Connection Machine CM-2. Symposium on Parallel Algorithms and Architectures, Hilton Head, SC. 3-16, July 1991.
5. Boncz, P.; Manegold, S.; Kersten, M.; Database Architecture Optimized for new Bottleneck: Memory Access; International Conference on Very Large Databases (VLDB); 1999.
6. Cha, S.; Sangyong, H; Kim, K.; Kwon, K.; Cache-Conscious Concurrency Control of Main-Memory Indexes on Shared-Memory Multiprocessor Systems; International Conference on Very Large Databases (VLDB); 2001.
7. Site visited in april 27th 2004: http://www.embarcadero.com/resources/tech_papers/WhatPerformanceDoINeed_6_6.pdf
8. M. Fayad, D. Schmidt, R. Johnson. Building Application Frameworks: OO Foundations of Framework Design. John Wiley and Sons, 1999.
9. Christos Faloutsos, Raymond T. Ng, Timos K. Sellis: Flexible and Adaptable Buffer Management Techniques for Database Management Systems. IEEE Transactions on Computers 44(4): 546-560 (1995).
10. Gamma, E. ; Helm, R.; Johnson, R.; Vlissades J.; Design Patterns: Elements of Reusable Software Architecture. Addison-Wesley, 1995.
11. Manegold, S.; Boncz, P.; Kersten, M.; What happens during a Join ? Dissecting CPU and Memory Optimization Effects; International Conference on Very Large Databases (VLDB); 2000.
12. Site visited in april 26th 2004: http://otn.oracle.com/products/rdb/pdf/rdb_7105_on_ev56.pdf
13. Site visited in april 27th 2004: http://www.tpc.org/
14. Object Management Group: UML Profile for Schedulability, Performance and Time. OMG Document ad/2001-06-14,http://www.omg.org; 2004.
15. Zhou, J.; Ross, K; Buffering Access to memory-resident index structures; International Conference on Very Large Databases (VLDB); 2003.
16. Dikaiakos, M.; Samaras, G.; A performance Analysis Framework for Mobile-Agent Systems; Workshop on Infrastructure for Scalable Multi-Agents Systems (ICAA); 2000.
17. Site visited in may 17th 2004:http://encyclopedia.thefreedictionary.com/Software+component
18. Boral, H.; DeWitt, D.; A Methodology for Database System Performance Evaluation; Proceedings of the 1984 SIGMOD Conference, June, 1984.
19. Petriu, D.; Shen, H.;Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications; in Computer Performance Evaluation - Modelling Techniques and Tools, (Tony Fields, Peter Harrison, Jeremy Bradley, Uli Harder, Eds.) Lecture Notes in Computer Science 2324, pp.159-177, Springer Verlag, 2002.
20. Site visited in may 17th 2004:http://linkage.rockefeller.edu/wli/zipf/

21. Copeland, G.; Khoshafian, S.; A decomposition storage model; in Proceedings of ACM SIGMOD Conference, pages 268-279, 1985.

22. Ramakrishnan, R.; Gehrke, J.; Database Management Systems; McGraw-Hill; 2 edition, 2000.