An Evaluation of Object Management System Architectures for Software Engineering Applications

Jayavel Shanmugasundaram, Barbara Staudt Lerner, Lori Clarke
Department of Computer Science
University of Massachusetts
Amherst, MA 01003 USA
+1 413 545 3787
{shan, lerner, clarke}@cs.umass.edu

ABSTRACT
Software engineering applications require sophisticated object management system support for creating and manipulating software objects. One of the key issues for object management systems is distribution. Addressing this issue in the context of software engineering applications is particularly challenging because they have widely varying object access profiles. Two fundamental approaches to dealing with distribution are the object server architecture, where objects are shipped to the application program, and the operation server architecture, where operation requests are shipped to where the objects reside. We compare these architectures experimentally to determine the conditions under which each performs better.

KEYWORDS
Distributed object management, experimental evaluation

1 INTRODUCTION

Software engineering applications create and manipulate software artifacts that tend to be large, inter-related collections of objects with many complex operations performed on them, sometimes over a long duration of time. One effective technique to aid in the creation and manipulation of such software artifacts is to use an underlying object management system that models objects as instances of abstract data types (ADTs) and provides features like persistence and concurrency control.

With the advent of global networking, software engineering activities are becoming increasingly distributed as can be witnessed in new and emerging applications such as collaborative software engineering (e.g., [2, 7, 9, 10]) and software processes (e.g., [1, 6]). As a result, one of the key issues to be addressed by object management systems for software engineering is distribution, where objects and the application programs accessing the objects are located on different machines, perhaps located at various geographical sites. Since accesses to remote objects incur a network overhead that is likely to be much more expensive than local communication overhead, the performance implications of a distributed object management system architecture need to be carefully evaluated.

There are two fundamental ways to deal with distribution. One way is to ship the remote objects to the application program so that the application program can perform operations on the object. The other way is to ship the operation requests from the application program to the place where the object resides, so that the operations can be performed on the object. These two approaches, which we call the *object server architecture* and the *operation server architecture* respectively, represent two ends in a spectrum of architectures for distributed object management systems. Most object oriented databases use an architecture similar to the object server architecture [3, 4] while most relational databases use an architecture similar to the operation server architecture [12].

Addressing this issue of distribution is particularly challenging in the context of software engineering applications because they have widely varying object access profiles, unlike most traditional database applications. For example, during design a software engineer might use a browser to search for an object containing a particular piece of the design from a relatively large collection of objects, resulting in a single comparison operation being applied to each object that is accessed until the matching object is found. In contrast, a static analysis

This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract F30602-94-C-0137. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory, or the U.S. Government.

Jayavel Shanmugasundaram's current address is Computer Sciences Department, University of Wisconsin - Madison, Madison, WI 53706 USA

tool might operate on a fine-grained representation of a program containing 10^6 nodes or more using algorithms that have quadratic complexity or worse. Given these different object access profiles, it seems unlikely that one distribution approach will perform well for both applications. On the other hand, it may be possible to predict which architecture will perform best for a particular application so that the appropriate architecture can be chosen.

While these two architectures are not new, there has been little work comparing the performance of the two architectures in order that such predictions can be made in a scientific manner. Most of the work comparing the architectures has been in the context of query optimization [5, 8]. In this paper, we evaluate the performance of the object server architecture and the operation server architecture and present an empirical model to aid in the selection of the appropriate architecture for a particular application. The rest of the paper is organized as follows. Section 2 describes the object management system model and introduces the object server and the operation server architectures. Section 3 presents a detailed performance study of these two architectures and identifies the conditions under which each architecture excels. Section 4 summarizes our results and discusses plans for future work.

2 THE OBJECT MANAGEMENT SYSTEM: MODEL AND ARCHITECTURE

In this section, we outline the object management system model that we assume in this paper and then describe a simple non-distributed architecture, the object server architecture, and the operation server architecture for such a system.

2.1 The Object Management System Model

In our object management system model, we assume that objects are instances of ADTs. The operations that can be applied to objects are specified as part of the ADT definition. We also assume that each object has a uniquely identifying object id. An object may refer to other objects by means of their object ids. An object referred to by other objects is called a *component object.*

2.2 Non-Distributed Object Management System Architecture

A simple non-distributed architecture of an object management system is shown in Figure 1. In the interest of clarity, we restrict our current discussion to the case where an application program accesses instances of just one ADT.

There are three main parts in the architecture, namely the ADT implementation, the object cache and the stor-

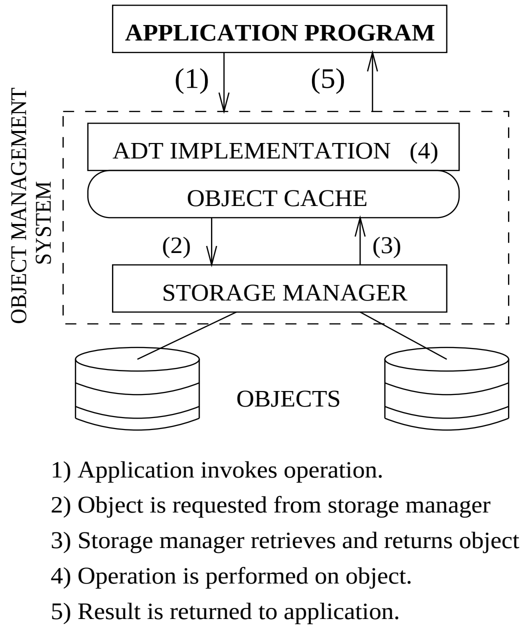

Figure 1: Non-Distributed Architecture

age manager. Application programs invoke operations on an object. If the object is not present in the object cache, a request is sent to the storage manager for the desired instance. The storage manager reads the object from the disk and returns it to the object cache. The invoked operation is then performed on the object present in the object cache. As part of the execution of this operation, the application may indirectly invoke operations on component objects. If these component objects are not present in the object cache, they are requested from the storage manager before operations can be performed on them. The result of the invoked operation is then returned to the application program.

A *session* is the unit of interaction between the application program and the object management system. All the modified objects in the object cache are flushed to the storage manager at the end of every session.

The non-distributed architecture was designed for the situation in which the different parts could interact using local procedure calls. In the case where application programs need to access remote objects, however, alternative architectures have to be explored.

2.3 The Object Server Architecture

In the object server architecture, the storage manager is the server and the remaining parts of the architecture reside with the application program in the client. The architecture is so named because the interaction between the client and the server is at the level of objects. This architecture is similar to the "data-shipping" architecture commonly used in object oriented database systems [3].

The key difference between the non-distributed architecture and the object server architecture is in the cost of sending objects to the client and returning modified objects to the server at the end of a session.

2.4 The Operation Server Architecture

In the operation server architecture, the storage manager and the ADT implementation form the server and the application program becomes the client. The architecture is so named because the interaction between the client and the server is at the level of operations on objects. This architecture is similar to the "query-shipping" architecture used in relational database systems.

The key difference between the non-distributed architecture and the operation server architecture is in the cost of sending operation requests to the server and returning the results to the client.

3 PERFORMANCE STUDY OF THE ARCHITECTURES

In this section, we present the results of a study comparing the performance of the operations within a single session using the non-distributed, object server, and operation server architectures. We first identify the factors that affect the performance of architectures and then study their effects experimentally.

3.1 Factors of Applications Affecting Performance

The five main steps listed in Figure 1 are the same for all architectures. These steps give rise to the following factors that are likely to vary significantly from application to application. The factors derived from the application as well as the communication costs derived from the architectures determine the overall performance of an application. The empirical model employs both the application factors and communication costs to predict application performance as described in Section 3.2. In the following list of application factors, we show in italics the identifier that represents each factor in the empirical model.

- **Number of operations** invoked directly on an object by the application program (*num_of_ops*).
- **Number of objects retrieved** by the storage manager by an operation invoked (directly or indirectly) by the application program (*num_of_objs*).
- **Size of object** operated on by an operation invoked (directly or indirectly) by the application program (*obj_size*).
- **Time to execute** an operation performed directly on a local object, exclusive of the time to execute operation calls indirectly on component objects (*time_to_execute*).

The meaning of the factors is somewhat subtle and thus requires further explanation.

Number of operations. The client-server communication for the operation server involves the sending of operation requests from the application to the object management system and the returning of results to the application. As a result, increasing the number of operation requests stresses this communication and decreases the expected performance of the application. Only the requests going from the application program to the ADT implementation incur this communication overhead. The requests from the ADT implementation to component objects are local procedure calls that occur entirely on the server. As a result, the experiment varies the number of operations from the application program with the expectation that the more operations that are performed on a single object, the worse the operation server architecture will perform compared to the object server architecture.

We do not consider the size of parameters to an operation and the size of the return result of an operation as factors. In an object oriented environment, we expect most operations to take few parameters and these parameters will either be object ids or simple scalar types like integers. Similarly, we expect the return results of operations to be object ids or simple scalar types. Thus, the size of the parameters and the size of the return results of operations are not expected to vary widely from application to application.

Number of objects retrieved and size of objects. The client-server communication for the object server architecture involves sending objects from the storage manager to the ADT implementation. Thus, the factors in an application program that will affect the communication costs for the object server architecture are the number of objects retrieved by an operation and the size of the objects. Since the ADT implementation resides at the client, all objects manipulated during the execution of an application program must be shipped to the client. As a result, the number of objects retrieved includes the objects retrieved directly by operation calls made from the application program as well as the component objects retrieved by operation calls made by the ADT implementation (and thus indirectly from the application program). The experiment used only read operations. To include write operations, we would also need to add a factor to represent the number of objects modified during a session since these objects must be sent back to the storage manager at the conclusion of a session.

Time to execute. The time that it takes to execute an operation depends upon the number of objects that need to be shipped to the client during its execution in the case of the object server, and the number of operation calls made to component objects in the case of the operation server. Thus, measuring the raw time of the operations would have been redundant with the other factors that we were measuring. To avoid this redundancy, the time that we measured is the time spent in an operation minus the time spent retrieving objects and the time spent in operation calls on component objects. The time to execute an operation therefore roughly equates to the architecture-independent complexity of the operation. Time to execute will vary as the power and load on the client and server machines vary. In our experimentation we used equivalently-powered and similarly-loaded machines for the client and server and therefore expected the time to execute an operation to be the same on both architectures. As we later describe, we were surprised to find that complex operations performed worse on the operation server architecture than on the object server architecture even under these conditions.

3.2 A Simple Empirical Model

We model the time required for performing operations in an architecture by determining the time required for performing the five steps outlined in Figure 1. The time for each step depends on the application factors listed above as well as client-server communication costs, which are architecture dependent. This leads to an empirical model that can be used to predict the expected performance of an application given the expected values for the factors and the measured values of the architecture-dependent costs. The time for each step is modeled as follows:

- **Operation invocation by the application program**. The time required for performing this step is characterized by the formula:

  $$op\_invocation\_time \times num\_of\_ops$$

  where *op_invocation_time* is the time for *a single* operation invocation by the application program. We expect *op_invocation_time* to be significantly higher for the operation server architecture than for the other architectures.

- **Requests to the storage manager**. The time required for making requests for objects (directly or indirectly accessed by an operation) is characterized by:

  $$storage\_mgr\_request\_time \times num\_of\_objs$$

  where *storage_mgr_request_time* is the time required for *a single* request message to be sent to the storage manager. We expect this time to be significantly higher for the object server architecture than for the other architectures.

- **Retrieving and returning objects from the storage manager**. The time for retrieving objects and sending them to the object cache is represented by the formula:

  $$(retrieval\_overhead + send\_overhead + (retrieval\_time + send\_time) \times obj\_size) \times num\_of\_objs$$

  where *retrieval_overhead* is the overhead for a retrieval operation in the storage manager and *retrieval_time* is the time to retrieve an object of unit size (excluding the overhead). Similarly, *send_overhead* is the overhead for sending an object from the storage manager to the object cache and *send_time* is the time to send an object of unit size (excluding the overhead). We expect the values of *retrieval_overhead* and *retrieval_time* to be approximately the same for all the architectures. However, we expect the values of *send_overhead* and *send_time* to be higher for the object server architecture than for the other architectures.

- **Performing operations on objects**. The formula given below characterizes the time for performing operations on objects.

  $$time\_to\_execute \times num\_of\_objs \times num\_of\_ops$$

  Since all the variables given above are application factors, we expect the time required for performing operations on objects to be approximately the same for all the architectures, assuming equally powerful and equally loaded clients and servers.

- **Returning results to the application program**. The time required for returning the results of operations to the application programs is given by the formula:

  $$result\_return\_time \times num\_of\_ops$$

  where *result_return_time* is the time required for returning the results of *a single* operation to the application program. We expect *result_return_time* to be higher for the operation server architecture than the other architectures.

Combining the time required for all steps detailed above, we arrive at the following formula for the performance of operations in an architecture within a session:

$op\_communication\_cost \times num\_of\_ops +$
$constant\_storage\_mgr\_cost \times num\_of\_objs +$
$variable\_storage\_mgr\_cost \times obj\_size \times num\_of\_objs +$
$time\_to\_execute \times num\_of\_objs \times num\_of\_ops$

where $op\_communication\_cost = op\_invocation\_time + result\_return\_time$, $constant\_storage\_mgr\_cost$ is the storage manager overhead that is independent of object size which equates to $storage\_mgr\_request\_time + retrieval\_overhead + send\_overhead$, and $variable\_storage\_mgr\_cost$ is the storage manager cost that varies with object size which equates to $retrieval\_time + send\_time$. From the discussion above, we expect $op\_communication\_cost$ to be higher for the operation server architecture than for the object server architecture and $constant\_storage\_mgr\_cost$ and $variable\_storage\_mgr\_cost$ to be higher for the object server architecture than for the operation server architecture. This illustrates the basic tradeoff between the operation server architecture and the object server architecture - the operation server architecture is expected to perform better when an operation accesses many different objects while the object server architecture is expected to perform better when there are many operations invoked on the same object.

3.3 Experimental Setup

For the purpose of this evaluation, we used Pleiades [14], an ADT based object management system, as the underlying object management system and modified it to implement the object server and operation server architectures by replacing local procedure calls with the appropriate remote procedure calls. The results of this evaluation, however, are not tied to Pleiades and should be applicable to any ADT based object management system satisfying the model described in Section 2.1.

For this performance evaluation, we investigated the effects of the factors outlined in Section 3.1 on the performance of operations within a session. For the purpose of keeping the experimentation tractable, we designed the experiment such that

- The cache is large enough to hold all objects needed during a session.
- All entries in the object cache are invalidated at the end of a session.
- Objects are brought over one at a time from the storage manager to the object cache.
- All operations are read operations.
- During a session of the application program:
  - Each invocation of an operation on an object is deterministic.
  - Each object has the same size.
  - The time required to execute an operation locally on each object, exclusive of operation calls on component objects, is roughly constant.

The experimentation involved running applications with different values for the factors identified earlier. We selected values that one might reasonably expect software engineering applications to possess. All applications used list traversal as their main operation. The number of operations invoked was controlled by changing the number of times the traversal operation was invoked during a session of the application program. This was varied from 1 to 2001 in steps of 100. The number of objects retrieved was varied by changing the length of the list from 1 to 391 in steps of 30. The size of the object was varied by changing the size of each entry in the list from 50 bytes to 3122 bytes, in steps of 1024 bytes. The time to perform an operation on each object was varied by changing the complexity of the operation performed on each object during the traversal. This took on the values 1, 2, 4 and 6, where an operation with complexity value of 1 was a very simple operation on an ADT instance requiring a few 100s of machine instructions to execute. Operations with a higher complexity value, say x, had to execute approximately x times as many instructions as the operation with a complexity value of 1.

The experiment was performed on DEC Alpha workstations, with a clock speed of 166 MHz, running Digital UNIX V3.2C. For the non-distributed architecture, all components of the architecture resided on the same machine. In the case of both the object server and the operation server architectures, the client and server components resided on different machines. The machines were connected using a 10 Mbps ethernet LAN and the client and server machines communicated using Q remote procedure calls [11]. All experiments were performed when there were no other user processes running on the machines and when there was little network traffic. Each experiment was repeated four times and the average time for the last three runs was used as the measured time. This was done in order to warm the processor cache during the first run and to eliminate any noise during the subsequent three runs. All timing was performed using the Unix *time* command.

3.4 Experimental Results

In this section, we present selected performance results[1] of the non-distributed, object server, and operation server architectures using the experimental setup de-

[1] More detailed results can be found at http://laser.cs.umass.edu/dot-experiment.html.

scribed above. As expected, the non-distributed architecture always outperforms the other architectures and so we do not discuss that architecture further.

Effect of Number of Operations. As expected, the operation server performs better than the object server when the number of operations performed is low. Figure 2 compares the performance of operations under the three architectures when the number of objects is 181, the object size is 50 bytes and the operation complexity is 1. With this configuration, the object server performs better after about 800 operations are executed during the session, meaning that the cost of retrieving an object whose size is 50 bytes and performing the operation in the client is approximately 4.4 times the cost of invoking the operation remotely and performing the operation on the server for operation complexity 1. The crossover point at which the object server does better varies depending on the object size and operation complexity as will be shown later.

Effect of Number of Objects. As expected, the object server performs better when the number of objects retrieved is low. Figure 5 compares the performance of operations under the three architectures when the number of operations is fixed at 1001 and the operation complexity is 1 but the number of objects varies. Here, the operation server architecture does better when each operation retrieves 200 or more objects. The crossover point again varies depending on operation complexity and object size as is shown later.

Note that the performance of the operation server architecture is roughly constant when the number of operations is large and the number of objects is small. We believe that this is due to the fact that each call to the operation server results in scheduling and process switching overhead. When operations execute very quickly, the server is waiting longer for an opportunity to execute than it actually spends in execution, resulting in a measurable impact on server performance. As the execution time increases, the impact of this overhead lessens as is demonstrated here by increasing the number of objects and later in Figure 6 by increasing operation complexity.

Effect of Operation Complexity. We expected operation complexity to degrade the performance of the operation server and object server equally since we used equally-powered and similarly-loaded machines for the client and the server. The experimental results, however, indicated that operation complexity has a bigger impact on the operation server. Figure 3 compares the performance of operations under the three architectures when the number of objects is 181 and the operation complexity is 4. Comparing this to Figure 2, we see that the crossover point moved from 800 operations to 600 operations as the complexity increased from 1 to 4. Figure 6 compares the performance of operations under the three architectures when the number of operations is fixed at 1001 and the operation complexity is 4. In this case, the object server architecture always outperforms the operation server architecture. This contrasts with the results from Figure 5 where the operation complexity was 1, and the operation server architecture performed better when there were more than 200 objects.

Effect of Object Size. As expected, operation server performance improves relative to object server performance as the size of the objects increases, since this increases the client/server costs for the object server architecture. Figure 4 compares the performance of operations under the three architectures when the number of objects is 181, the operation complexity is 1 and the size of an object is 2098 bytes. The operation server architecture always performs better than the object server architecture at this object size. Compare this with Figure 2 where the object size was 50 bytes. In that case the object server architecture did better only when more than 800 operations were applied to the objects. Similarly, Figure 7 compares the performance of operations under the three architectures when the number of operations is fixed at 1001, the operation complexity is 1 and the size of an object is 2098 bytes. Here the object server architecture outperforms the operation server architecture only when fewer than 10 objects are retrieved. In contrast, Figure 5, where the object size was 50 bytes, performed better until 180 objects were retrieved.

Application Performance. As per the empirical model discussed in section 3.2, we expect the performance of applications in the architectures to vary as per the following formula:

$$op\_communication\_cost \times num\_of\_ops + constant\_storage\_mgr\_cost \times num\_of\_objs + variable\_storage\_mgr\_cost \times obj\_size \times num\_of\_objs + time\_to\_execute \times num\_of\_objs \times num\_of\_ops$$

Table 1 shows the values for the unknowns for the different architectures obtained using the least mean square approximation technique when the object size is 50 bytes and the operation complexity is 1.

From the above results, we can see that the empirical model fits the experimental results very well, particularly for the non-distributed and object server architectures. The standard deviation is higher for the operation server because the data fits a plane less well due to the flatness observed when operations execute very quickly, as described earlier. As expected, the value of $op\_communication\_cost$ is much higher for the operation server architecture than the object server architecture. Similarly, the storage manager cost ($constant\_storage\_mgr\_cost$ + 50 $\times$

Unknown	Non-Distributed Architecture	Object Server Architecture	Operation Server Architecture
Op communication cost	1.448 ms	1.683 ms	7.534 ms
Storage mgr cost	0.153 ms	18.899 ms	0.337 ms
Time to execute	0.016 ms	0.014 ms	0.008 ms
Standard deviation	0.05 s	0.1 s	0.7 s

Table 1: Values for Architecture Factors: Object size = 50 bytes, Operation complexity = 1

variable_storage_mgr_cost) is much higher for the object server architectures. We believe the disparity in *time_to_execute* for the operation server architecture is again due to the relatively poor fit of the data to a plane.

The operation communication cost and storage manager cost are virtually unaffected by operation complexity changes, as we expected. Table 2 shows the change in time to execute for the different operation complexities. For the non-distributed and object server architectures the time to execute increases linearly with the operation complexity. This is not true for the operation server, however, because the overhead of scheduling and process switching has less effect at higher operation complexities (when the time spent performing the operation dominates scheduling time) and thus the empirical model fits the experimental results better at higher operation complexities. Also, note that the time to execute operations of a given operation complexity differs for the architectures and influences their performance. In particular, even on an unloaded operation server, the time to execute anything other than trivial operations is slower than with the other architectures due to resource contention with the storage manager and Q.

When we fit the empirical model to the performance results for each architecture for different object sizes, the operation communication cost and time to execute are virtually unchanged, as we expected. Table 3 shows the changes in the storage manager cost as the object size is varied, demonstrating how the object server architecture quickly degrades in performance as object size increases.

Experimental Conclusions. The experiment confirmed our hypothesis that the object server architecture would perform better when many operations were applied to a small collection of objects while the operation server architecture would perform better when a few operations were applied to a large collection of objects. Before developing the empirical model and performing the experiment, however, we did not know exactly where the tradeoffs would be. The results of our experiment indicate that object size has a tremendous impact on the performance of the object server. Indeed, it seems that the object server architecture should only be considered in the cases where objects are small or where operations have high complexity. Even in these situations, it is important to take into consideration the number of objects that would need to be transferred and weigh the cost of transferring those objects against the cost of potentially many operation calls on the server. On the other hand, as contention for the server increases, it seems the object server architecture will become increasingly attractive since it reduces the load on the server, although we have not measured this effect experimentally.

Returning to our earlier example, static analysis tools use a large number of small objects and an algorithm with high complexity. In this case, the cost of executing a polynomial number of operations using the operation server is higher than sending n small objects to the client using the object server. In contrast, the searching algorithm is low complexity, and involves potentially many large objects. In this case, it is better to perform the search at the server rather than send all the objects to the client when most of them will never be used again. The empirical model would facilitate the prediction of the appropriate architecture for applications with less extreme object access profiles.

4 CONCLUSIONS AND FUTURE WORK

Software engineering applications have varying object access profiles that influence the selection of the appropriate object management system architecture. In this paper, we have studied and empirically modeled the performance of two fundamental architectures to support distribution, namely the the object server and the operation server architectures. The performance study shows that, as expected, the object server architecture performs better when many operations are performed on the same object and that the operation server architecture performs better when an operation accesses many objects. Further, an increase in the size of the objects significantly degrades the performance of the object server architecture but not that of the operation server architecture. The empirical model developed in this paper helps to quantitatively determine the conditions under which one architecture performs better than the other. Such an empirical model is likely to be very useful in practice because in some applications, we ex-

Complexity	Non-Distributed Architecture	Object Server Architecture	Operation Server Architecture
1	0.016 ms	0.014 ms	0.008 ms
2	0.032 ms	0.029 ms	0.029 ms
4	0.065 ms	0.059 ms	0.068 ms
6	0.096 ms	0.085 ms	0.107 ms

Table 2: Values for Time to Execute: Varying Complexity of Operations, Object size = 50 bytes

Object Size	Non-Distributed Architecture	Object Server Architecture	Operation Server Architecture
50	0.153 ms	18.899 ms	0.337 ms
1074	0.276 ms	224.282 ms	0.423 ms
2098	0.371 ms	411.577 ms	0.579 ms
3122	0.435 ms	604.342 ms	0.823 ms

Table 3: Values for Storage Manager Cost: Varying Object Size, Operation complexity = 1

pect that selecting the faster architecture would reduce application program running time considerably.

Although the results presented in this paper help in determining an appropriate architecture for a given application, much work remains to be done in this area. The empirical model should be extended to model the performance of the architectures at session boundaries and for applications that have many sessions. The empirical model also needs to be validated using actual application programs. Further, the empirical model does not currently take into account the effects of caching strategies. Sophisticated caching strategies such as page level caching of objects, where each page could have more than one object [3], and caching objects in the object cache between sessions could be used in the object server architecture. Operation result caching, where the return value of a read only operation is cached at the client for future use [12], could be used for the operation server architecture. The usage of these caching strategies may significantly affect the tradeoffs between the two architectures and thus needs to be explored.

One of the long term goals of this work is to tailor an appropriate architecture for each application, possibly at run-time and on a per object basis, taking into account a wide variety of issues. In this context, the exploration of hybrid architectures that combine features of the object and operation server seems to be a step in the right direction. We are also interested in examining how well these architectures support additional functionality such as collaboration, heterogeneity, type evolution, recovery, and resilience to failure.

ACKNOWLEDGEMENTS

We would like to thank Jagan Peri for helping with the implementation of the object server architecture and Peri Tarr for her insights into the Pleiades implementation, assistance in designing the experiments, and comments on the paper.

REFERENCES

[1] V. Ambriola, R. Conradi, A. Fuggetta. Assessing process-centered software engineering environments. *ACM Transactions on Software Engineering and Methodology*, 6(3):283–328, (July 1997).

[2] S. Bandinelli, E. Di Nitto, A. Fuggetta. Supporting cooperation in the SPADE-1 Environment. *IEEE Transactions on Software Engineering*, 22(12):9–20, (December 1996).

[3] D. DeWitt, P. Futtersack, D. Maier, F. Velez. A study of three alternative workstation-server architectures for object-oriented database systems. *Proceedings of the 16th VLDB Conference*, Brisbane, Australia, August 1990.

[4] M. Franklin. *Client Data Caching*, Kluwer Academic Press, Boston, 1996.

[5] M. Franklin, B. T. Jónsson, D. Kossmann. Performance tradeoffs for client-server query processing. *Proceedings of the SIGMOD Conference*, pages 149–160, June 1996.

[6] A. Fuggetta, A. Wolf, eds. *Trends in Software: Software Process*, Volume 4, John Wiley and Sons, New York, 1996.

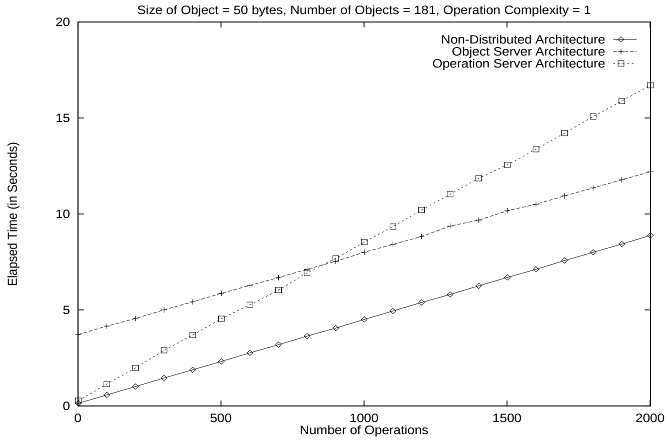

Figure 2: Varying Number of Operations: Object size = 50 bytes, Operation complexity = 1

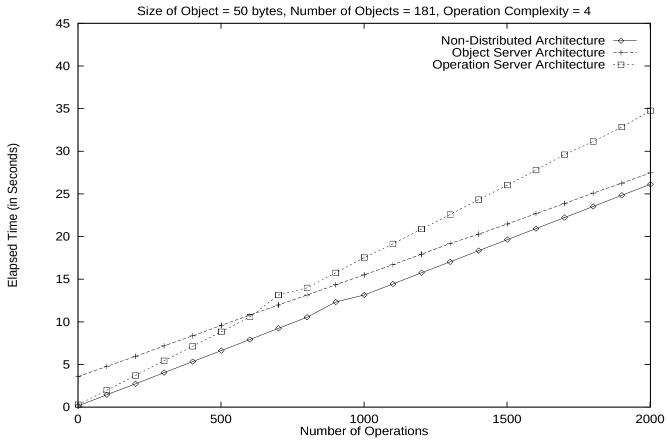

Figure 3: Varying Number of Operations: Object size = 50 bytes, Operation complexity = 4

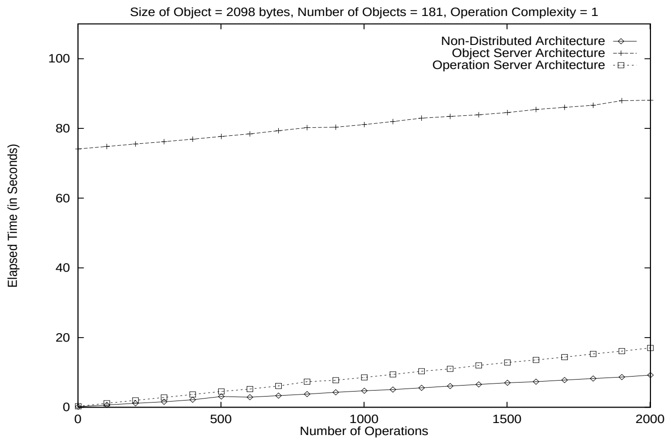

Figure 4: Varying Number of Operations: Object size = 2098 bytes, Operation complexity = 1

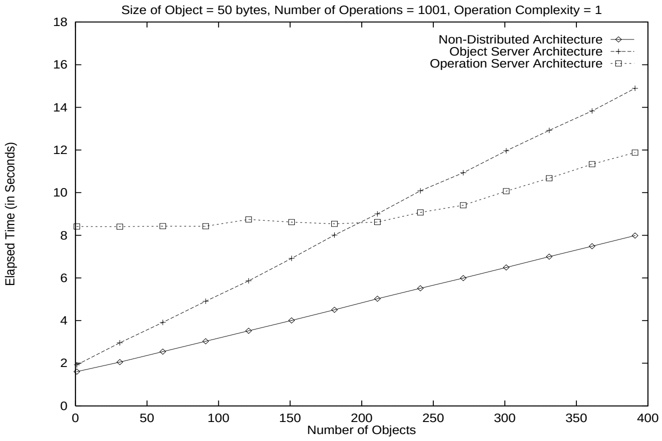

Figure 5: Varying Number of Objects: Object size = 50 bytes, Operation complexity = 1

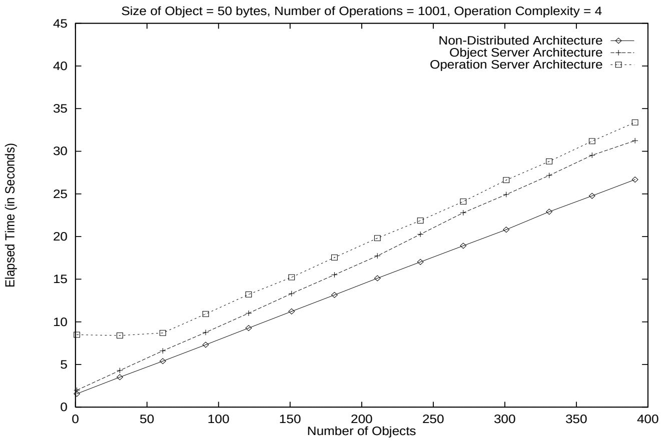

Figure 6: Varying Number of Objects: Object size = 50 bytes, Operation complexity = 4

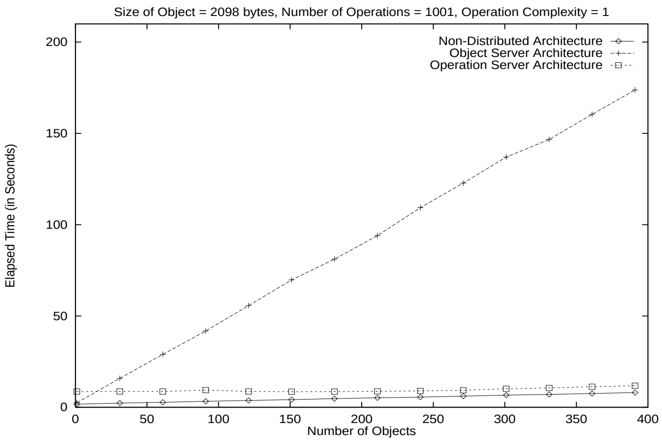

Figure 7: Varying Number of Objects: Object size = 2098 bytes, Operation complexity = 1

[7] W. H. Harrison, H. Ossher, P. F. Sweeney. Coordinating concurrent development. *Proceedings of the Conference on Computer-Supported Cooperative Work*, pages 157–168, Los Angeles, California, 1990.

[8] B. P. Jenq, D. Woelk, W. Kim, W. Lee. Query processing in distributed ORION. *Proceedings of the International Conference on Extending Database Technology*, pages 169–187, Venice, Italy, March 1990.

[9] G. Junkerman, B. Peuschel, W. Schäfer, S. Wolf. MERLIN: Supporting cooperation in software development through a knowledge-based environment. *Software Process Modeling and Technology*, Research Studies Press Ltd, 1994.

[10] S. M. Kaplan, W. J. Tolone, A. M. Carrol, D. P. Bogia, C. Bignoli, Supporting collaborative software development with Conversation Builder. *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*, December 1992.

[11] M. J. Maybee, D. M. Heimbigner, L. J. Osterweil. Multilanguage interoperability in distributed systems. *Proceedings of the 18th International Conference on Software Engineering*, pages 451–463, Berlin, Germany, March 1996.

[12] N. Roussopoulos, C. M. Chen, S. Kelley, A. Delis, Y. Papakonstantinou. The ADMS project: Views R Us. *IEEE Conference on Data Engineering*, pages 19–28, March 1995.

[13] M. Stonebraker, P. M. Aoki, R. Devine, W. Litwin, M. A. Olson. Mariposa: A new architecture for distributed data. *IEEE Data Engineering Conference*, pages 54–65, February 1994.

[14] P. L. Tarr, L. A. Clarke. PLEIADES: An object management system for software engineering environments. *ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 56–70, Los Angeles, December 1993.