Harnessing Replication in Object Allocation

H. T. Barney, and G. C. Low

Abstract—The design of distributed systems involves the partitioning of the system into components or partitions and the allocation of these components to physical nodes. Techniques have been proposed for both the partitioning and allocation process. However these techniques suffer from a number of limitations. For instance object replication has the potential to greatly improve the performance of an object orientated distributed system but can be difficult to use effectively and there are few techniques that support the developer in harnessing object replication.

This paper presents a methodological technique that helps developers decide how objects should be allocated in order to improve performance in a distributed system that supports replication. The performance of the proposed technique is demonstrated and tested on an example system.

Keywords— Allocation, Distributed Systems, Replication.

I. INTRODUCTION

WHILE there are a number of object oriented techniques for allocating the components of a distributed system [1-3], each has its limitations. The focus of this paper is the development of an allocation technique that supports object replication. Allocation techniques from object oriented distributed systems and distributed databases are examined before a modified technique is proposed. A worked example demonstrates the new technique.

II. LITERATURE REVIEW

A system that is to be distributed around a network must be broken down into components that are allocated to physical nodes. The process of breaking the system down into components is called "partitioning". The process of allocating the components (partitions) around the network is called "allocation". The allocation process usually has the goal of minimizing inter-process communication cost, minimizing execution cost, load balancing, increasing system reliability and providing scalability [4].

A. Object Oriented Distributed System Allocation Techniques

1) Low and Rasmussen's allocation technique[1]

Low and Rasmussen consider both the communication costs between partitions and the processing load on each of the nodes in the distributed system. They then use a heuristic

algorithm [5] to merge partitions until an adequate allocation arrangement is reached.

This technique does not horizontally fragment instances of classes. Consequently all objects that reside in the same partition are allocated to the same node. Applying replication at the granularity of a whole class of objects may mean that there will be little benefit derived from applying the technique. If there are a large number of objects in a particular class then the cost of replicating and keeping replicas current for the whole class will almost always outweigh any benefit derived from locality of invocation.

This allocation technique also lacks support when deciding which objects should be replicated or how they should be replicated.

2) Chang and Tseng's allocation technique[2]

Chang and Tseng's allocation technique makes allocation decisions at an object not a class level so their technique does not have the problems associated with allocating whole classes to individual nodes encountered in [1]. Although allocation in this technique can take place at the granularity of individual instances of classes, their technique offers no guidance on how to model the interactions between individual instances. It is therefore not clear how employing this method will support the developer in allocating different objects. In a large system with hundreds or even thousands of instances of each class, modeling the interactions between these instances is an insurmountable task.

3) Purao et. al's allocation technique[3]

Purao et al describe a method for allocation that uses a series of formulae to model different aspects of the performance of a distributed system. The four formulae estimate: the match of the fragments to their respective processors, the communication volume, concurrency potential and the cost of maintaining a given set of replicas. The user of this methodology must provide information about the network upon which the system is to be allocated and details about the design of the system. Additionally, horizontal fragmentation criteria for all classes must be provided beforehand.

Multiple possible allocation arrangements are produced that are locally, but not globally, optimal. The user can then choose which of the suggested solutions has the performance characteristics that most closely match the desired performance characteristics of the final system. The chosen solution is then used to seed the next iteration of the technique to produce a further set of locally optimal solutions clustered around the chosen solution. This process continues until the user is satisfied that the allocation arrangement produced satisfactorily meets the desired performance characteristics.

The suggested approach has two main drawbacks: it does not consider processor loads and fragmentation decisions are

made before the allocation process starts. The cost in terms of CPU time is not a factor considered in Purao et al.'s model of the distributed system and as such their technique may produce an allocation arrangement where the CPU of one or more of the nodes is overloaded making it a bottle-neck for the entire system.

The fact that horizontal fragmentation criteria must be decided before the allocation process can commence may involve unnecessary work if all the instances of the given class end up being allocated to a single node. It may also be difficult to determine meaningful horizontal fragmentation criteria a priori without some information about the context of the decision, and what the fragmentation of the instances of that class aims to achieve.

B. Database Approaches

Most distributed database approaches to the allocation problem split the allocation process into two stages, fragmentation and allocation of these fragments. Fragmentation is the process of dividing a relation into meaningful segments. These fragments are then allocated to nodes in the distributed system. The allocation processes advocated by [6] and [7] will be examined to judge their suitability for application to object orientated distributed systems.

It is noted that there are some object oriented database techniques for partitioning such as [8] but these techniques generally focus on "address partitioning for efficient access of pages from secondary memory and require information that may not be easily available during the design stage" [3] and as such are not of direct interest for the purposes of this paper.

1) Ceri and Pelgatti's allocation technique [6]

Ceri and Pelgatti offer a process whereby fragmentation and allocation are performed as independent exercises. Their process requires the creation of predicates that are used to divide the relations.

Their allocation process determines the optimal allocation of the predetermined database fragments based on where and how they are accessed using a 0,1 integer programming approach. This approach attempts to minimize the estimated communication costs between network nodes This process is, however, by their own estimation, "very simplistic", and does not incorporate "the relationship between fragments…. the cost of integrity enforcement… concurrency enforcement". Despite the simplifications made to produce this equation this formulation of the problem has also been proven to be NP-complete, making it impractical to use in a case where there are a large number of nodes and fragments.

2) Tamhankar and Ram's allocation technique[7]

Tamhankar and Ram present an integrated fragmentation and allocation technique for distributed databases. They identified seven criteria that a system designer can use to determine the fragmentation, replication and allocation strategy for each relation. The characteristics suggested for analysis are: the site of the updates, cost of updates, sites of queries, volume of data, data currency requirements and other overriding considerations.

A developer analyses all the relations in the system using the above criteria. An initial guess at the fragmentation and replication strategy is decided based on these characteristics. They produced a table of recommended fragmentation and replication strategies based on these seven criteria. The strategy can be modified in a process called secondary distribution to meet individual design goals. There are three secondary distribution stages: response time, availability and storage space.

III. PROPOSED TECHNIQUE

None of the reviewed allocation techniques adequately solves the allocation problem for object-oriented systems where replication is available. Given the evaluation of the various allocation techniques and the analysis of their applicability to the field of object-oriented systems a new technique, incorporating the advantages of the reviewed allocation techniques, is proposed.

Tamhankar and Ram's approach has many advantages. It incorporates processor loads and replication. Additionally, fragmentation predicates are only created where necessary and it provides guidance on which type of predicate is suitable given the characteristics of a given partition. In these two respects it offers an advantage over the other approaches examined.

However, their approach cannot be directly applied to an object oriented system because all the metrics and the many of the specifics of their allocation heuristics are specific to databases. The metrics suggested by Low and Rasmussen [1], which are valid in the context of an object oriented system, can be used in conjunction with Tamhankar and Ram's approach to produce a new heuristic approach to the allocation problem that better meets the goals for the allocation process [4].

In line with Tamhankar and Ram, the proposed technique is composed of three stages: primary distribution, secondary distribution for response time and secondary distribution for storage space.

A. Primary Distributions

Primary distribution is an initial attempt at fragmentation and allocation. The decision about fragmentation and allocation is based on seven criteria: sites of updates, cost of updates, sites of queries, cost of queries, volume of data, currency of data and any overriding considerations that may affect the allocation of that partition. The following variables will be of interest when estimating various aspects of the distributed system's performance.

The following interpretation of seven criteria for primary distribution within the context of object-oriented systems is suggested.

LIST OF TERMS

X	The node to be examined.
Cost$_x$	Component communications cost for node to be examined. It includes the communications cost and associated processor overhead associated with the communication.
$\forall i; i \in E$	All events, **i**, from the set of events **E**.
n$_j$	The number of times the event **j** is repeated in a period of interest.
$\forall j; k \leftrightarrow l, k \in$ x	All messages, **j**, where object **l** requests a service from **k** and **k** is an object residing on node **x**
m$_j$	The number of times the message, **j**, is sent between **k** and **l** in one occurrence of event **i**.
loc$_j$	The number of lines of code needed to implement the service requested by message **j**.
C$_j$	Cost of requesting a service provided by **k** requested by **l** by message **j** in the event **i**. For example [1] recommend incorporating an additional cost of 3 units (equivalent lines of code) for each method invocation. This allows details, like a reference to the calling object, which method is being invoked etc, to be incorporated.
u$_j$	This variable is 1 if message j will trigger an update and 0 if the message does not trigger an update.
Storage$_x$	The storage space required to store the partition, **x**.
d$_i$	The average amount of data for each instance of the object **i**.
i$_i$	The expected number of instances of object **i**.
s$_i$	The amount of space required for the executable portion of the object **i**.

1) Sites of update (SU)

Whether the updates to a partition's constituent objects originate from a single site (1) or multiple sites (M). Only the original source of an event is to be considered as the site of an update, even when those updates originate from other objects

2) Cost of update (CU)

Whether the cost of the updates that occur to a partition's constituent objects is high (HI) or low (LO). Partitions are designated as having a HI or LO cost depending on the total cost of updates to that partition.

An existing component communications cost model for distributed object oriented systems [1] is adapted to differentiate between updates, u_j and queries, $1-u_j$. Thus the following equation gives the cost model for updates.

$$Cost_x = \sum_{\forall i: i \in E}\left(n_i \times \sum_{\forall j: k \leftrightarrow l, k \in x}\left[m_j u_j \left(loc_j + c_j \right) \right] \right) \quad (1)$$

Equation (1) should be used to estimate the cost of updates for each partition. Partitions should be ordered by the cost of updates. The group of partitions that form the bottom 10% of the total should be designated as LO. Those partitions comprising the remaining 90% should be designated as having HI update costs.

3) Sites of queries (SQ)

As for sites of update, this indicates whether updates to the partition emanate from a single site (1) or multiple sites (M).

4) Cost of queries (CQ)

As for cost of updates except that the following cost equation should be used where only the cost of queries is considered.

$$Cost_x = \sum_{\forall i: i \in E}\left(n_i \times \sum_{\forall j: k \leftrightarrow l, k \in x}\left[m_j \left(1 - u_j \right) \left(loc_j + c_j \right) \right] \right) \quad (2)$$

5) Volume of data (HI/MD/LO)

The formula for determining storage space requirements is given below:

$$Storage_x = \sum_{\forall i: i \in x}\left(d_i \times i_i + s_i \right) \quad (3)$$

The proportion of the total storage space requirements each partition requires, estimated using Equation 3, can be used to guide the allocation of high, medium or low designations. Partitions should be ordered by their estimated storage requirements. The partitions that form the bottom 10% of the total are designated as LO, the next 20% are designated MD and the remaining partitions are designated as HI.

6) Currency of data (CD)

This criterion is used to indicate the kind of currency requirements the objects in that partition require. (O) if one day old, or older data is acceptable or (C) if any object in a partition requires current data.

7) Overriding considerations for a site (OC)

If there are any overriding considerations that have not been covered by the other criteria, like security, that will dictate the allocation of the partition then this criteria has a value of Y.

Tamhankar and Ram suggest a range of partitioning arrangements for primary distribution based on the above criteria. This can be used un-modified. Based on the additional information gathered in this process the developer may re-consider the partitioning arrangement originally decided upon.

B. Secondary Distribution for Response Time

Secondary distribution for response time improvement allows the designer to improve distribution of the application to reduce the cost of transactions. Improvements are then sought to the distribution arrangement by: relocating a fragment, maintaining a copy of the fragment and/or clustering fragments for a transaction.

This process requires the user of the technique to estimate the processor and network loads produced when an allocation arrangement resulting from the primary distribution phase is used. For the purposes of this paper the cost functions proposed in [1] were extended to incorporate support for replication. Description of this process is beyond the scope of the current paper.

C. Secondary Distribution for Storage Space

The final stage in the proposed allocation process is secondary distribution for storage space. This stage allows the incorporation of storage space constraints in the allocation arrangement. It is recommended that Equation 3 be used to estimate storage space requirements for each node and the re-distribution guidelines from Tamhankar and Ram [7] be used.

IV. WORKED EXAMPLE

The new allocation technique will be demonstrated using an

illustrative example.

A. Description of Example System

A group of fifty travel agents, have convinced the transport operators that they need a centralised booking system. The airlines and cruise companies, convinced of the cost savings centralised booking system would entail, agree to participate.

Travel Agents help customers find flights or cruises that suit their travel plans. They search for flights or cruises that will take a customer where he/she wants to go on the dates requested. When a customer has chosen a flight, the Travel Agent tries to book tickets for the customer. If a customer decides that he/she no longer wishes to take the flight or cruise the travel agent can cancel it.

The airlines have decided that they will offer a frequent flyer bonus points program. Customers that have flown with an airline will gain bonus points that the customer can later reclaim on flights with that airline. The cruise companies have decided that a similar frequent cruiser program would not benefit them and prefer to offer their customers slightly cheaper prices on their cruises instead.

There are five main use cases in this system: new customer, search for suitable flights, book ticket, cancel ticket and update frequent flyer/cruiser points.

In order to produce allocation arrangements using the allocation technique described above detailed descriptions of each of the use case scenarios, and the network upon which the system is to be to allocated are required.

Estimates for the implementation details for each class and the methods that they implement are also required. The number of instances of each object is estimated along with the average size of each instance and the methods required to implement that object.

Traces of each use case are required for each event. These traces include estimates of the following for each message exchanged between objects in the execution of the event: size of the message sent (method arguments or return arguments), the estimated cost of executing that message (number of lines of code is one suggested metric) and the number of times that message is repeated.

In order to allocate the aforementioned objects to actual nodes in the system, the layout of the nodes in the system must be understood. Fifty travel agencies take part in this system along with two cruise operators and three airlines. The Travel Agencies are based in capital cities situated on Australia's eastern seaboard, 20 in Sydney, 12 in Brisbane and the remaining 18 in Melbourne. The following table shows how the events are divided between the three cities.

The conglomerate decides to establish three nodes for this system. One based in Sydney to service the Sydney offices, a second in Brisbane and a third in Melbourne. Two of the airlines are based in Sydney while the third is based in Brisbane. The requests for updating frequent flyer points are assumed to originate from the location of the airline. One of the cruise operators is based in Sydney, the other in Melbourne.

TABLE I
NUMBER OF EVENTS ORIGINATING FROM EACH NODE

	Bris	Syd	Melb	Total
New Customer	24	40	36	100
Search	2,160	3,600	3,240	9,000
Book	720	1,200	1,080	3,000
Cancel Ticket	24	40	36	100
Update Bonus Points	1	2	0	3

B. Applying the New Allocation Technique

The classes must be partitioned before the allocation algorithm can be used. The final partitioning arrangement was produced according to the process described by Barney [9]:

A —Flight, Cruise, Route   **D** — Ticket
B — Customer, Bonus Account   **E** — Airline
C — Airport, Port   **F** — Travel Agent,

This partitioning arrangement reflects an analysis for communication costs, concurrency, replication concerns and class similarity.

The process for fragmentation and allocation of a system begins with primary distribution. These fragmentation and allocation decisions are gradually refined to optimise all aspects of system performance. Primary distribution incorporates: the overhead for update events, query events, the volume of data and the currency of data. The corresponding total instances of each class and the number of replicas (in brackets) are shown in Table II.

TABLE II
PRIMARY DISTRIBUTION EXAMPLE

	Class	Instances assigned to each node		
		Bris.	Syd.	Melb.
A	Route	25 (10)	30 (10)	25 (10)
	Flight	5,000 (2,000)	6000 (2000)	5000 (2000)
	Cruise	100 (40)	120 (40)	100 (40)
B	Customer	2,867 (0)	25,333 (0)	1,800 (0)
	Bonus A/C	1,667 (0)	58,333 (0)	0 (0)
C	Airport	0 (0)	230 (0)	0 (0)
	Port	0 (0)	70 (0)	0 (0)
D	Ticket	310000 (62000)	372,000 (62,000)	310,000 (62,000)
E	Airline	0 (0)	3 (0)	0 (0)
F	Travel Agent	0 (0)	50 (0)	(0)

Table III shows the estimated properties of each fragment in the dimensions discussed previously: site of the updates, cost of updates, sites of queries, volume of data, data currency requirements and other overriding considerations.

TABLE III
ESTIMATED FRAGMENT PROPERITES

	SU	CU	SQ	CQ	VD	CD	OC
A	1	LO	M	HI	LO	C	N
B	M	HI	M	LO	HI	O	N
C	M	LO	M	LO	LO	C	N
D	M	HI	M	HI	MD	C	N
E	M	LO	M	LO	LO	O	N
F	M	LO	M	LO	LO	O	N

1) Partition A - Flight, Cruise and Route

The suggested primary allocation strategy, using Table III, is horizontal fragmentation of the partition, combined with creating synchronised copies of some of those fragments.

In order to perform horizontal fragmentation it is recommend that a location based-attribute(s) that can be used to perform horizontal fragmentation be identified.

The destination list of the Route class is a suitable location-based attribute for horizontal fragmentation of this partition. People would be far more likely to book flights including the city they are currently in as part of that route. If a Route object exists on a node then all corresponding Flight or Cruise instances that service that route should also be located on the node.

2) Partition B - Customer and Bonus Account

The proposed technique recommends horizontal fragmentation for this partition if a location-based attribute can be identified. If no locality of reference can be identified then the relation should be allocated to the partition where the most updates take place.

Updates to the partition originate from the three airlines or from the Travel Agent when a new instance of the customer class is created. Two of the airlines are based in Sydney and the third in Brisbane. Customers, however, have a mix of bonus point accounts from the three different airlines. The following horizontal fragmentation arrangement is therefore suggested.

Instances of the Customer class that don't have any bonus point accounts can remain at the node upon which they were created.

Customers with only one bonus point account should be allocated to the node where the airline corresponding to the Bonus Point Account is located.

If the customer has more than one bonus point account at least one of those bonus accounts will be for an airline from Sydney as two airlines are located in Sydney and one in Brisbane. Therefore, partition fragments that match this criterion should be allocated to the Sydney node.

This arrangement is reflected in the following table.

TABLE IV
SUMMARY OF ALLOCATION ARRANGEMENT FOR PARTITION B

	Brisbane		Sydney		Melbourne	
	Cust	Bonus A/C	Cust	A/C	Cust.	Bonus A/C
Customer (0 A/Cs)	1200	0	2000	0	1800	0
Customer (1 A/C)	1667	1667	3333	3333	0	0
Customer (2+ A/Cs)	0	0	20000	55,000	0	0
Total	2867	1667	25333	58333	1800	0

3) Partition C - Airport and Port

The proposed technique recommends that the instances of these two classes be allocated to one of the sites of update. Most of the airlines and travel agents operate out of Sydney so the most updates to this class will come from users of the system located in Sydney. This partition will therefore be allocated exclusively to the Sydney node.

4) Partition D – Ticket

Horizontal partitioning using a location-based attribute is recommended for this partition. Furthermore, it is recommended that one copy of the fragment/partition with update synchronization should be allocated to the site with the highest volume of queries.

In the case of the Ticket object, its reference to the Transport object (parent class of both Cruise and Flight classes) acts as a valid attribute for deciding to which node the fragments of this partition should be allocated. The instances of the Ticket class should therefore be assigned to the node where the Ticket's corresponding Flight or Cruise instance has been allocated. In the case of the Flight or Cruise instances that have been replicated, the corresponding Ticket objects can also be replicated to all nodes where that flight object exists.

5) Partition E – Airline

The proposed technique recommends that the instances of this partition be allocated to one of the sites of updates. Two of the three airlines are located in Sydney and one in Brisbane. The majority of updates to instances of the Airline object will therefore originate in Sydney. It is therefore suggested that Sydney is the best location for this partition.

6) Partition F – Travel Agent

The proposed technique recommends allocating all instances of this partition one of the sites of updates. As most of the travel agents are located in Sydney this partition should be allocated to the Sydney node.

C. Secondary Distribution for Response Time

The results of performing performance analysis are summarized in Table V. It demonstrates quite clearly that the search event is the biggest contributor to both processor load and inter-process communication. This method should therefore be the focus of any changes to improve response time.

TABLE V
SECONDARY ANALYSIS FOR RESPONSE TIME

Event	Communication Costs		Processor Load	
	Total	As %	Total	As %
New Customer	79164	6%	36880	0%
Search	1286100	91%	713961000	96%
Book	24060	2%	26467140	4%
Cancel Ticket	150	0%	2700	0%
Update Bonus Points	20196	1%	2435070	0%
Total	1409670	100%	742902790	100%

The communication between the Travel Agent class and the Flight class represent the single biggest contribution to the inter-process communication volume in the Search event. Two potential changes that could be made to improve response time are fully replicating partitions A and F or to allocate a single copy of the A and F partitions to the Sydney node.

By having complete replicas of partitions A and F on every node, the inter-process communication between these classes would be converted into local communication, thus reducing both processor load and inter-process communication volumes. Creating replicas of these partitions will probably increase the communication and processor cost due to replication associated with the Cancel Ticket event and any other updates to partitions A and F. These updates would however be of low frequency compared to the frequency of the Search event. The Search event represents such an overwhelming proportion of the inter-process communication and processor load costs that the increase in load for this event would probably be more than offset by the reductions gained in the Search event.

Similarly, the alternative strategy of allocating partitions A and F to the Sydney node eliminates all inter-process communication between these two partitions in the Search event. This must be weighed against the concurrency lost and the higher load placed on the Sydney node if this allocation arrangement were chosen.

The implications of these, and other, alternatives must be evaluated in the light of the desired performance characteristics of the system being developed.

D. Storage Space Optimization

The technique suggested here also has scope for optimising the storage space requirements of the system.

Over 65% of the storage space required for the system is allocated to the Sydney node. If the amount of data stored in Sydney were to exceed the hardware limits there are a number of strategies that could be used to reduce the storage space required on that node.

Instances of the customer object represent approximately 60% of the storage space requirements of the system. Most of the instances of this class are located on the Sydney node. One strategy to reduce the storage space requirements on that node would be to relocate some of those instances of the customer object to Brisbane or Melbourne. This could impact negatively on the performance of the Create Customer and Tally Bonus Points events but since both of these events represent very small proportions of the total processor and inter-process communication loads this would probably have very little impact on overall system performance.

V. Comparison with Other Techniques

Preliminary comparison of the proposed technique with existing object allocation techniques have shown that the proposed technique is competitive with, and in some cases superior to, the other object allocation techniques examined. A more thorough comparison of the performance of the proposed technique and existing object allocation techniques is proposed as an important part of the future work necessary to validate the proposed allocation technique.

VI. Conclusion

Several allocation techniques were surveyed from not only the object orientated world but also databases and non-object orientated distributed systems. The database allocation technique produced by the technique described in [7] was found to best support the goals for the allocation process [4] by: supporting replication, minimizing inter-process communication and execution costs, providing support for scalability and fault tolerance and integrating allocation and fragmentation into a single process. The alternative techniques examined implemented, at best, a sub-set of these advantages. The Tamhankar and Ram technique was adapted for use with object oriented systems.

This was achieved by: establishing a mapping between the database and object orientated terminologies and modifying their technique to account for differences between databases and object orientated systems.

The proposed technique was applied to an illustrative example. The new technique produced an allocation arrangement that meets the goals set for the allocation process by Shatz and Wang [4]. By incorporating, processor load estimates, object replication and partition fragmentation this technique is more comprehensive than the alternatives considered in the domain of object oriented distributed systems when object replication is available.

In order to fully validate the proposed technique's effectiveness, further empirical evaluation of its performance is necessary. A comparison of the performance of the proposed technique and of the other allocation techniques is an important part of the future work that needs to be undertaken.

References

[1] G. C. Low and G. Rasmussen, "Partitioning and Allocation of Objects in Distributed Application Development," Journal of Research and Practice on Information Technology,, vol. 32, pp. 75-106, 2000.

[2] W. T. Chang and C. C. Tseng, "Clustering Approach to Grouping Objects in Message-Passing Systems," Journal of Object Orientated Programming, vol. 7, pp. 42-43, 46-50, 1995.

[3] S. Purao, H. K. Jain, and D. L. Nazareth, "Exploiting Design Information to Derive Object Distribution Models," IEEE Transactions on Systems, Man, and Cybernetics, vol. 32, pp. 320-334, 2002.

[4] S. M. Shatz, Development of Distributed Software: Concepts and Tools: Macmillan Publishing, 1993.

[5] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," IEEE Computer, vol. 15, pp. 50 - 6, 1982.

[6] S. Ceri and P. G., Distributed Databases: Principles and Systems: Mc Graw-Hill, 1984.

[7] A. M. Tamhankar and S. Ram, "Database Fragmentation and Allocation: An Integrated Technique and Case Study," IEEE Transactions on Systems, Man, and Cybernetics, vol. 28, pp. 288-305, 1998.

[8] K. Karlapalem and Q. Li, "A Framework for Class Partitioning in Object-Oriented Databases," Distributed and Parallel Databases, vol. 8, pp. 333-366, 2000.

[9] H. Barney, "Object Replication: a methodology for improving the performance of object oriented systems," in School of Information Systems, Technology and Management, vol. BSc. (Hons). Sydney: UNSW, 2003, pp. 205.