Competitive Analysis of Caching in Distributed Databases*

Ouri Wolfson and Yixiu Huang
Electrical Engineering and Computer Science Department
University of Illinois
Chicago, Illinois 60607

*A preliminary version of this paper has appeared in the Proceedings of the 10th International Conference on Data Engineering, Feburary 1994, Houston, Texas. This research was supported in part by Grants NSF IRI-9224605, NSF IRI-9408750, DARPA N66001-97-2-8901, NATO CRG-960648, AFSOR F49620-93-1-0059.

Abstract

This paper makes two contributions. First, we introduce a model for evaluating the performance of data allocation and replication algorithms in distributed databases. The model is comprehensive in the sense that it accounts for I/O cost, for communication cost, and, because of reliability considerations, for limits on the minimum number of copies of the object. The model captures existing replica-management algorithms, such as read-one-write-all, quorum-consensus, etc. These algorithms are static in the sense that, in the absence of failures, the copies of each object are allocated to a fixed set of processors.

In modern distributed databases, particularly in mobile computing environments, processors will dynamically store objects in their local database and will relinquish them. Therefore, as a second contribution of this paper, we introduce an algorithm for automatic dynamic allocation of replicas to processors. Then, using the new model, we compare the performance of the traditional read-one-write-all static allocation algorithm, to the performance of the dynamic allocation algorithm. As a result, we obtain the relationship between the communication cost and I/O cost for which static allocation is superior to dynamic allocation, and the relationships for which dynamic allocation is superior.

Key words: *object allocation, caching, distributed database, mobile computing, wireless communication, competitiveness.*

1. Introduction

In a distributed database an object is usually replicated in the local database of multiple processors for performance and availability. The object is accessed, i.e. read and written, from multiple geographically distributed locations.

In this paper we analyze the cost of servicing a set of read-write requests for a replicated object. This set of requests is usually ordered by some concurrency-control mechanism, such that each read request accesses the most recent version of the object (i.e. the version written by the most recent write request). The cost of servicing a read or write request depends on the *allocation scheme* of the object, namely the set of processors that store the most recent version of the object in their local databases. For example, if a reading processor is in the allocation scheme, then the read can be satisfied locally; otherwise it must be satisfied from a remote processor, and it becomes more expensive.

The allocation scheme of an object is either dynamic or static, namely it either changes as the read-write requests are executed, or it remains fixed. The reason for changing the allocation scheme is that the larger the allocation scheme the smaller the cost of an average read-request, and the bigger the cost of an average write request. Thus, in a read-intensive environment a large allocation scheme is mandated, whereas in a write-intensive environment a small allocation scheme is mandated. Caching is a particular form of dynamic allocation in which a processor that reads an object saves a copy of that object, and thus it joins the allocation scheme. In this paper we study caching in a peer-to-peer (rather than client server) environment.

The main results of this paper concern caching, however we discuss this in the larger context of dynamic allocation. More specifically, we discuss dynamic allocation performed by Distributed Object Management (DOM) algorithms. A DOM algorithm maps each request to a set of processors that execute the request, and it determines the allocation scheme of the object at any point in time. We distinguish between online and offline DOM algorithms. An offline DOM algorithm, when servicing a request, has a priori knowledge of all the future requests. An online algorithm does not do so. In this paper we compare the cost of static and dynamic online algorithms. We do so by a worst-case comparison of an online algorithm to an optimal, ideal offline algorithm.

Our worst case analysis of replication is performed in two cost models. In one, the *stationary-computing* (SC) model, each read/write request has a cost in terms of both communication and I/O. For example, a write request of the object usually results in the transmission of the update to multiple processors that store copies of the object; this involves communication cost. Furthermore, each one of these processors outputs

the object to its local database; this involves an I/O cost.

In the *mobile-computing* (MC) model, the cost of read and write requests represents a dollar value for communication. The wireless network provider usually charges a fee per wireless message, and this fee depends on the length of the message. For example, RAM Mobile Data Co. charges 4 cents for a two-character message, and 12 cents for a 450-character message. Therefore the cost of a write is the dollar cost expended to transmit the write to the mobile computers caching replicas of the object.

The rest of the paper is organized as follows. In section 2 we introduce a dynamic online algorithm, and summarize the results of this paper. In section 3 we present the model, we formalize the cost function, and we define distributed object management algorithms. In section 4 we define the static and dynamic online algorithms in terms of the proposed model, and we analyze them. In section 5 we discuss practical issues (e.g. failures) concerning dynamic allocation. We also discuss extensions of the model beyond reads and writes. In section 6 we compare our work with relevant literature. Finally, in section 7 we summarize and discuss our results.

2. Results Summary

In this paper we achieve two objectives. First, we propose a model and a methodology to analyze and compare online DOM algorithms. Basically, the proposed methodology to show that DOM algorithm A is *superior* to DOM algorithm B, is as follows. First, show that A is α-competitive, i.e., there exists a constant $\alpha > 1$ such that for any sequence s of read-write requests: (the cost of A on s) $\leq \alpha \times$ (the cost of the optimal offline DOM algorithm on s). Then, show that B is not α-competitive, i.e., that there are sequences of requests for which (the cost of B) $> \alpha \times$ (the cost of the optimal offline DOM algorithm). This means that in the worst case the cost of A is lower than the cost of B. [1]

The second objective of this paper is to introduce a new DOM algorithm, namely Dynamic Allocation (DA), and to compare it with the traditional read-one-write-all Static Allocation (SA) DOM algorithm. The comparison is performed using the methodology discussed above. Both the SA and the DA algorithms are on-line, in the sense that they do not know the sequence of the requests a priori.

[1] This type of worst case analysis is useful in real-time systems that have to work under worst-case assumptions. It is useful when, for example, the objective is to avoid overloading/crashing a distributed system of a given capacity. Such overloading/crashing may occur when the I/O or communication queues suddenly begin to grow unboundedly bringing the system to its knees. An algorithm with a better worst-case behavior has a smaller chance of creating such an overload condition.

In addition to performance, a DOM algorithm is also affected by availability considerations. We assume that the SA and DA algorithms are subject to the following availability constraint: "For some integer t which is greater than 1, the allocation scheme must be of size which is at least t". In other words, t represents the minimum number of copies that must exist in the system. If the maximum number of processors that are expected to be "down" at any point in time is d, and availability dictates that at least one copy of the object be "up", then t is set to $d+1$.

In subsection 2.1 we present and discuss the SA and DA algorithms, and in subsection 2.2 we summarize the results of the comparison between them.

2.1 Algorithms description

In the SA and DA algorithms, a set of processors act as servers of an object o. Even so, we do not assume a client-server environment. The algorithms can be incorporated in a peer-to-peer system in which the set of servers differs from object to object.

In a mobile computing environment the servers may be mobile. Indeed, with mobile computers becoming more reliable and better connected, it may be reasonable to have the whole database distributed among mobile computers.

Basically, DA differs from SA in the fact that it uses caching. More specifically, in DA a client caches the object that it reads; any write invalidates the cached copies.

Static Allocation: At all times, SA keeps a fixed allocation scheme Q which is of size t. All the processors in the system know which are the processors of Q. SA performs read-one-write-all. Namely, in response to a write request issued by a processor p, SA sends the object from p to each one of the processors in Q. In turn, each processor of Q outputs the object in its local database. In response to a read request issued by a processor p, SA requests a copy of the object from some processor $y \in Q$; in turn, y retrieves the replica from its local database and sends it to p. □

Dynamic Allocation: The DA algorithm receives as parameters a set F of $t-1$ processors, and a processor p that is not in F. The processors of F are called the *servers*, and p is called the *floating* processor. All the processors in the system know the id of the processors in $F \cup \{p\}$. The initial allocation scheme consists of $F \cup \{p\}$. Subsequently, at any point in time all the servers are in the allocation scheme, and at least one additional processor is there as well; however, the floating processor is not necessarily in the allocation scheme. So, for example, for nonserver nonfloating processors q and r, $F \cup \{q\} \cup \{r\}$ is a possible allocation scheme at some point in time.

The DA algorithm services read and write requests as follows. A read request from a processor of the allocation scheme is satisfied by inputing the object from the local database. A read request from a processor r outside the allocation scheme is satisfied by requesting a copy of the object from some server processor u; r saves the object in its local database (thus joining the allocation scheme), and u remembers that r is in the current allocation scheme by entering r in u's "join-list". The *join-list* of u consists of the set of processors that have read the object from u since the latest write.

A write request from some processor q outputs the object to the local database at q, and sends it to all the servers; then each server outputs the object in its local database. If q is a server, then q also sends a copy of the object to the floating processor (in order to satisfy the availability constraint). [2] Additionally, the write request results in the invalidation of the copies of the object at all the other processors (since their version is obsolete). This is done as follows. Each server, upon receiving the write, sends an 'invalidate' control-messages to the processors in its "join-list"; except that, obviously, if q is in some join list, the invalidation message is not sent to q. To summarize the effects of a write, consider the allocation scheme A immediately after a write from a processor q. If q is in F then $A = F \cup \{p\}$, and if q is not in F then $A = F \cup \{q\}$. $\square$

The DA algorithm differs slightly from the standard caching-with-write-invalidation in a client-server system. The difference is that in addition to the processors in F, i.e. the server processors, at every point in time there are other processors in the allocation scheme. However, the other processors are not fixed, and they vary depending on the read-write pattern. The reason for this difference from a client server system is that in a peer to peer system it is more efficient to make a writing processor a member of the allocation scheme.

The SA and DA algorithms may be incorporated in a transaction-oriented system that ensures serializability by using a concurrency control mechanism, such as two-phase-locking. In this case a copy of the object is locked before being read, written, or invalidated.

2.2 Cost Comparison of the SA and DA Algorithms

Assume that the I/O cost of servicing an access request, i.e. the cost of inputting (outputting) the object from (to) the local database, is c_{io}. This means that the average I/O cost of a database read or write in the distributed system is c_{io}. This does not mean that every read and every write results in an I/O to disk. For example, if buffering in the distributed system is such that on average one in every 10 reads/writes results

[2] Observe that if q is not a server, then the availability constraint is satisfied since q has the latest version of the object.

in a disk access, then the I/O cost of a read or a write is 1/10th the cost of a disk access. The cost of a disk access is the additional load that the operation puts on the distributed system. For example, if an I/O operation takes one time unit and the distributed system can perform 10,000 I/O's per time unit, then the cost of a disk access is 1/10000.

In terms of communication, there are two types of messages associated with servicing access requests, *control-messages* and *data-messages*. An example of a control message is a request message, in which a processor p (that does not have a copy of the object) requests another processor q to transfer a copy of the object to p. A data message is a message in which the object is transmitted between processors. Control messages are usually much shorter than data messages, therefore, different costs are associated with the two types of messages. The communication cost of a control-message is c_c, and the communication costs of a data-message is c_d.

In section 6 we will discuss how the DA algorithm copes with failures, but the cost-comparison results, discussed next, are obtained for the SA and DA algorithms operating in the normal mode, namely, in the absence of failures. For simplicity, in the stationary-computing model we normalize the cost by taking c_{io}=1. This means that c_c is the ratio of the cost of transmitting a control message to the I/O cost of a read-write request. Similarly, c_d is the ratio of the cost of transmitting a data message to the I/O cost of a read-write request.[3]

In the stationary-computing model we obtain the following results. We show that the SA algorithm is $(1+c_c+c_d)$-competitive. Then we show that the DA algorithm is $(2+2 \cdot c_c)$-competitive in general, and that the DA algorithm is $(2+c_c)$-competitive when $c_d > 1$. Interestingly, these competitiveness factors are independent of the integer t which limits the minimum number of copies in the system. Then we show that the SA algorithm is not γ-competitive for any $\gamma < 1 + c_c + c_d$. Therefore, when $c_d - 1 > 0$ (i.e., when the cost of a data-message is higher than the I/O cost) the DA algorithm is superior to the SA algorithm, since in this case $1 + c_c + c_d > 2 + c_c$. Then we show that the DA algorithm is not 1.5-competitive. Therefore, when $c_d + c_c < 0.5$ the SA algorithm is superior to the DA algorithm, since in this case $1 + c_c + c_d < 1.5$.

The results of the comparison are summarized in figure 1. The figure indicates the area on the c_c and c_d plane for which the SA algorithm is superior, and the area for which the DA algorithm is superior. The

[3]Observe that this formulation ignores storage cost. This is justified when storage is abundant, e.g., when each node has enough capacity to store a copy of each object in the database. When this is not the case it may be possible to include the storage cost in the I/O cost as follows. If on average the cost of storing a copy of the object for a time unit is 1/50000, and on average there is one local database I/O per time unit, then the I/O cost increases by 1/50000.

area in which $c_c > c_d$ is marked "Cannot be true", since a data message cannot be less costly than a control message. The reason for this is that the control message includes the object-id and operation (read, write, or invalidate) fields only, whereas the data message includes the object-id and operation (read, write, or invalidate) fields, as well as the object content. The area marked "Unknown" represents the c_c and c_d values for which it is currently unknown whether the DA algorithm is superior to the SA algorithm or vice versa. The reason for this uncertainty is the gap between the upper and lower bound on the competitiveness of the DA algorithm.

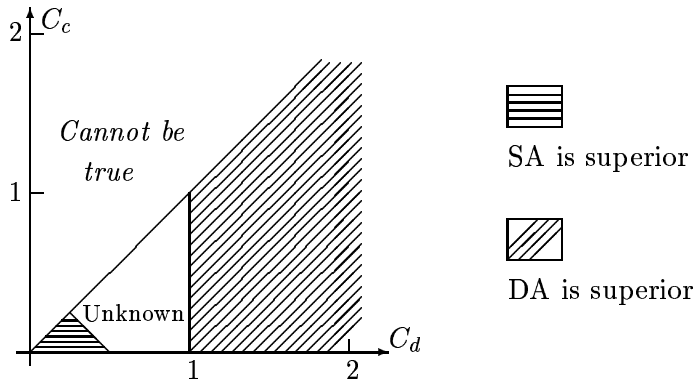

Figure 1.
Stationary-computing cost model

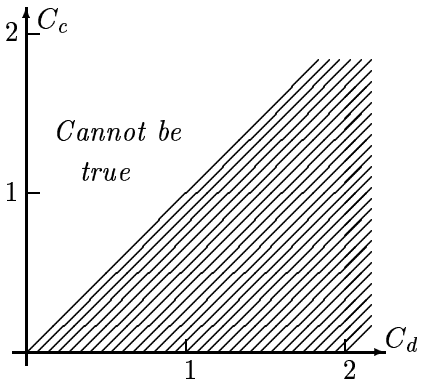

Figure 2.
Mobile-computing cost model

Then we consider the MC model. In this model the I/O cost is zero, the data-message cost is c_d and the control-message cost is c_c. We show that in a mobile computing environment the DA algorithm is $(2 + 3 \cdot \frac{c_c}{c_d})$-competitive, whereas the SA algorithm is no longer competitive. Thus, in such an environment the DA algorithm is strictly superior to the SA algorithm. Figure 2 illustrates this dominance of DA over SA.

3. The Model

3.1 Schedules and Allocation Schemes

A *distributed system* is a set of interconnected processors. The *local database* at a processor is a set of objects at the processor. Transactions operate on the object by issuing read and write requests. In this paper we address the allocation of a single object. A *schedule* is a finite sequence of read-write requests to the object, each of which is issued by a processor. For example, $\psi_0 = w^2 r^4 w^3 r^1 r^2$ is a schedule in which the first request is a write from processor 2, the second request is a read from processor 4, etc.

In practice, any pair of writes, or a read and a write, are totally ordered in a schedule, however, reads

can execute concurrently. Our analysis using the model applies almost verbatim even if reads between two consecutive writes are partially ordered.

Each write request in a schedule creats a new *version of the object*. Given a schedule, the *latest version of the object at a request* q is the version created by the most recent write request that precedes q.

Each request is mapped to a set of processors, namely the *execution set* of the request. Intuitively, for a write request the execution set is the set of processors which output the written object to their local database. For a read request the execution set is the set of processors from which the object is retrieved to satisfy the request.[4]

An *execution schedule* is a schedule of requests, each with its associated execution set. For example, $\xi_0 = w^2\{2,3\}$, $r^4\{1,2\}$, $w^3\{2,3\}$, $r^1\{1,2\}$, $r^2\{2\}$ is an execution schedule of ψ_0, where $\{2,3\}$ is the execution set of the first request w^2, $\{1,2\}$ is the execution set of the second request r^4, etc.

A read request brings the object to main memory of the processor that issued the read request, namely the *reading processor*. In case of dynamic allocation, the reading processor, s, may also store the object in its local database, in order to enable future reads at s to be local. A read request that results in saving the object in the local database is called a *saving-read*.

An *allocation schedule* is an execution schedule in which some reads are converted into saving-reads. A saving-read is denoted by an underlined read in the allocation schedule. For example, by converting the fourth request in ξ_0 into a saving-read, we obtain an allocation schedule $\phi_0 = w^2\{2,3\}$, $r^4\{1,2\}$, $w^3\{2,3\}$, $\underline{r}^1\{1,2\}$, $r^2\{2\}$. Observe that at the end of this allocation schedule the object is stored in the local databases of processors $\{1,2,3\}$.

The *initial allocation scheme* is a set of processors. Intuitively, it is the set of processors that have the object in their local database before the schedule begins. Given an allocation schedule, the *allocation scheme at a request* q is the set of processors that have the latest version of the object in the local database right before q is executed, but after the immediately-preceding request is executed. The allocation scheme at the first request is the initial allocation scheme. For example, consider the allocation schedule ϕ_0 above, and the initial allocation scheme $\{3,4\}$. The allocation scheme at the first request w^2 is $\{3,4\}$; the allocation scheme at the second, third, and fourth requests is $\{2,3\}$; the allocation scheme at the fifth request is $\{1,2,3\}$. We define a *data processor at a request* to be a processor which belongs to the allocation scheme at the request,

[4]Generally, a read request does not necessarily access a single copy of the object. For example, in quorum consensus (see [17, 30]), a read request retrieves a number of copies that have a read-quorum (and then discards all of them, except the one with the most recent time-stamp).

and a *non-data processor at a request* to be a processor which is not a data processor at the request.

In the introduction we mentioned the t-availability constraint. Formally, for an integer t, an allocation schedule satisfies the *t-availability constraint* if the allocation scheme at every request is of size which is at least t. For simplicity we shall assume that t is at least 2. We shall also assume that t is smaller than the total number of processors in the network, otherwise each write must be propagated to all the processors of the network, and the problems addressed in this paper become trivial.

Next we define a legal allocation schedule. Intuitively, it is an allocation schedule in which the execution set of every read request contains at least one data processor. In other words, the read request accesses the object in some processor that has a latest version of the object in its local database. Formally, given an initial allocation scheme, a *legal allocation schedule* is an allocation schedule in which the execution set of every read request has a non-empty intersection with the allocation scheme at the read request. For example, the allocation schedule ϕ_0 above is legal. However, ϕ_0 will be illegal if we change the execution set of the last request r^2 from $\{2\}$ to $\{4\}$.

A legal allocation schedule ϕ and a schedule ψ *correspond* to each other if ψ is obtained from ϕ by eliminating the execution sets, and turning every saving-read to a read. For example, ϕ_0 and ψ_0 correspond to each other.

3.2 The Cost of Requests in Stationary-Computing

We define the cost of inputting/outputting the object to the local database at a processor to be one unit. We denote by c_d the cost of transmitting a data-message from one processor to another, and we denote by c_c the cost of transmitting a control-message from one processor to another. Observe that this definition assumes a homogeneous system, in which the data-message between every pair of processors costs c_d, the control-message between every pair of processors costs c_c, and the I/O cost is identical at all the processors.

Given an allocation schedule, the cost of a request q, denoted $COST(q)$, is defined as follows. Suppose that $q = r^i$, i.e., q is a non-saving-read from processor i, and suppose that X is its execution set. In other words, the read request retrieves the object from the local database at the processors in X. If $i \in X$, then $COST(r^i) = (|X| - 1) \cdot c_c + |X| + (|X| - 1) \cdot c_d$. Intuitively, $(|X| - 1) \cdot c_c$ is the cost of submitting a read request to $(|X| - 1)$ processors, $|X|$ is the I/O cost of inputing the object from the local database at $|X|$ processors, and $(|X| - 1) \cdot c_d$ is the communication cost of sending the object from $(|X| - 1)$ processors to i. Otherwise, i.e., if $i \notin X$, then $COST(r^i) = |X| \cdot (c_c + 1 + c_d)$.

Suppose that $q = \underline{r}^i$, i.e., q is a saving-read from processor i, and suppose that X is its execution set.

Then,

$$COST(\underline{r}^i) = \begin{cases} (|X|-1)\cdot c_c + |X| + (|X|-1)\cdot c_d + 1 & \text{if } i \in X, \\ |X|\cdot(c_c + 1 + c_d) + 1 & \text{otherwise} \end{cases}$$

The cost of a saving-read is higher (by 1) than that of a non-saving read, to account for the extra I/O cost to save the object in the local database at i.

Suppose that $q = w^i$, i.e., q is a write request from processor i, suppose that X is its execution set, and suppose that Y is the allocation scheme at w^i. Then,

$$COST(w^i) = \begin{cases} |Y/X|\cdot c_c + (|X|-1)\cdot c_d + |X| & \text{if } i \in X, \\ |Y/X/\{i\}|\cdot c_c + |X|\cdot(c_d + 1) & \text{otherwise} \end{cases}$$

The explanation for the write cost is as follows. As part of servicing the write request that creates a new allocation scheme of the object, an 'invalidate' control message has to be sent to all the processors of Y/X, i.e. the processors at which the copy of the object is obsolete[5]; these are the processors of Y (the old allocation scheme) that are not in X (the new allocation scheme). Thus, the first term in the write-cost. The other terms are the cost of communicating the object to the processors of the new allocation scheme, and the cost of outputting the object to the local database at these processors.

For an allocation schedule $\xi = o_1 X_1 \ldots o_n X_n$ and an initial allocation scheme I, where o_i is either a read or a saving-read or a write, X_i is the execution set of o_i, we define the *cost of the allocation schedule* ξ, denoted by $COST(I, \xi)$, to be the sum of all costs of the read-write requests in the schedule, i.e.,

$$COST(I, \xi) = \sum_{i=1}^{n} COST(o_i)$$

3.3 The Cost of Requests in Mobile-Computing

Assume that the cost of inputting/outputting the object to/from the local database is zero. We still assume that the data message cost is c_d, and the control message cost is c_c.

Given an allocation schedule, suppose that o^i is a request in the schedule, Y is the allocation scheme at o^i, and X is its execution set. The cost of request o^i, denoted by $COST(o^i)$, is defined as follows.

$$COST(o^i) = \begin{cases} (|X|-1)\cdot(c_c + c_d) & \text{if } o \text{ is a read, and } i \in X \\ |X|\cdot(c_c + c_d) & \text{if } o \text{ is a read, and } i \notin X \\ |Y/X|\cdot c_c + (|X|-1)\cdot c_d & \text{if } o \text{ is a write, and } i \in X \\ |Y/X/\{i\}|\cdot c_c + |X|\cdot c_d & \text{if } o \text{ is a write, and } i \notin X \end{cases}$$

[5] X/Y denotes set-difference, to distinguish from arithmetic difference. Thus X/Y is the set of processors in X but not Y.

The reasoning for this cost function is identical to the one in the previous subsection. Observe that the cost of a read request executed only locally is zero. Observe also that the cost of a saving-read does not differ from that of a non-saving read.

3.4 Distributed Object Management Algorithms

A *distributed object management* (DOM) algorithm (say A) is an algorithm which, given a schedule ψ and an initial allocation scheme, produces a corresponding legal allocation schedule. We call this legal allocation schedule the *A-allocated schedule* of ψ and denote it by $las_A(\psi)$. For an integer $t > 1$, a DOM algorithm is *t-available constrained* if every legal allocation schedule that it produces satisfies the t-available constraint.

We define the *cost of the algorithm A on the schedule* ψ with initial allocation scheme I to be the cost of the A-allocated schedule, i.e., $COST(I, las_A(\psi))$, and sometime we denote it by[6] $COST_A(I, \psi)$. A t-available constrained DOM algorithm is *cost-optimal* if, for every schedule and for every initial allocation scheme, the cost of the legal allocation schedule that it produces is minimum among all the corresponding legal allocation schedules that satisfy the t-available constraint.

Now we define a special type of DOM algorithm, called an on-line DOM algorithm. An online DOM algorithm consists of a sequence of online steps. An *online step* takes as input an initial allocation scheme I, a legal allocation schedule ϕ, and a request q; it produces a new legal allocation schedule η that consists of the prefix ϕ followed by q (that could be underlined) with an execution set associated with q. Intuitively, an online step appends q to ϕ, associates an execution set with q, and, if q is a read, possibly turns it into a saving-read.

An *online DOM* algorithm is one that produces the legal allocation schedule (las) by "feeding" the requests of the schedule, sequentially, into an online step. Specifically, it provides the online step with the initial allocation scheme, the *las* produced in the previous invocation of the online step, and the next request in the sequence. For example, given an initial allocation scheme I, the SA online algorithm produces a legal allocation schedule by feeding the requests of a schedule into the following online step, that we call *Static Allocation Online Step* (SAOS). SAOS ignores the input legal allocation schedule, it associates with every write the execution set I, and it associates with every read a singleton that is some member in I. Intuitively, every read request accesses the object at some processor of I, and every write request is propagated to all

[6]Notice that if ψ_s is a sub-schedule of the schedule ψ, ξ_s is a sub-schedule of ξ and ξ_s corresponds to ψ_s, and the allocation scheme at the first request of ξ_s is I_s, then the A-allocated schedule with the initial allocation scheme I_s, $las_A(\xi_s)$, may not be ξ_s. Hence $COST(I_s, \xi_s)$ may not equal to $COST_A(I_s, \psi_s)$.

processors of I.

An *offline DOM* algorithm is one that is not online, namely, it can consider future requests when determining the execution set of the current one.

4. Comparison of On-Line Algorithms

In this section we define competitiveness as a performance measure for online DOM algorithms, then we present the static and dynamic on-line DOM algorithms, and then we analyze them in terms of competitiveness. The SA and DA algorithms defined in the introduction are t-available constrained, where t is the size of the initial allocation scheme. In other words, the SA and DA algorithms defined in the introduction become t-available constrained DOM's by providing them with an initial allocation scheme of size t.

4.1 Competitiveness

Competitiveness is a widely accepted way to measure the performance of an on-line algorithm (see [4, 10, 24]). Intuitively, an α-competitive online DOM algorithm is one which, for any schedule, costs at most α times as much as the minimum cost. Formally, a t-available constrained DOM algorithm, A, is α-competitive if there are two constants β and α, such that: for an arbitrary initial allocation scheme I and an arbitrary schedule ψ, $COST_A(I, \psi) \leq \alpha \cdot COST_{OPT}(I, \psi) + \beta$, where OPT is an offline t-available constrained DOM algorithm that produces the minimum cost legal allocation schedule for any input. We call α the *competitiveness factor* of the algorithm A.

4.2 Static and Dynamic Allocation In stationary Computing

4.2.1 The Static Allocation Algorithm

The SA algorithm was presented in the introduction. In terms of our model, SA maps a schedule into the following legal allocation schedule. Suppose that the initial allocation scheme is Q (of size t).

SA Algorithm

1. For a read request r^i, if $i \in Q$, then SA associates with it the execution set $\{i\}$. Otherwise, i.e., if $i \notin Q$, SA associates[7] with it the execution set that consists of an arbitrary processor in Q.
2. SA associates with every write the execution set Q.

[7]Remember that SA maps a schedule into a legal allocation schedule, i.e. by definition, it associates an execution set with each request.

Theorem 1: For any integer t, the SA algorithm is $(1 + c_c + c_d)$-competitive.

Proof : For any input schedule ψ, we split it into several sub-schedules. For each one of these sub-schedules, we will show that the incurred cost of SA is at most $(1 + c_c + c_d)$ times as much as the incurred cost of OPT. Thus we conclude the theorem.

Suppose that the schedule $\psi = \psi_0\psi_1 \ldots \psi_k$, where ψ_0 is zero or more reads which comes before the first write, and ψ_i is a write followed by zero or more reads for $1 \leq i \leq k$.

First, let's consider ψ_0. Suppose that $\psi_0 = r^{j_1}r^{j_2} \ldots r^{j_m}$. Each read costs at least one for the I/O, hence $COST_{OPT}(Q, \psi_0) \geq m$. On the other hand, a read in the SA-allocated schedule won't cost more than $(1 + c_c + c_d)$ since SA turns no read into a saving-read. Therefore $COST_{SA}(Q, \psi_0) \leq (1 + c_c + c_d) \cdot m \leq (1 + c_c + c_d) \cdot COST_{OPT}(Q, \psi_0)$.

Next we consider the subschedule ψ_i. Suppose that $\psi_i = w^{j_0}r^{j_1}r^{j_2} \ldots r^{j_m}$. For the OPT algorithm, suppose that Q_i is the allocation scheme at the i^{th} write of the OPT-allocated schedule. The write costs at least $(t - 1) \cdot c_d$ for communication and t for I/O, and the reads cost at least m for I/O's. Hence $COST_{OPT}(Q_i, \psi_i) \geq (t-1) \cdot c_d + t + m \geq t + m$. For the SA algorithm, the write in the SA-allocated schedule costs at most $t \cdot (1 + c_d)$ since the allocation scheme never changes, and each read costs at most $(1 + c_c + c_d)$, therefore $COST_{SA}(Q, \psi_i) \leq t \cdot (1 + c_d) + m \cdot (1 + c_c + c_d) \leq (t + m) \cdot (1 + c_c + c_d) \leq (1 + c_c + c_d) \cdot COST_{OPT}(Q_i, \psi_i)$.

□

The competitiveness factor of SA is *tight* in the following sense. For any constants γ $(< 1 + c_c + c_d)$, and $\beta \geq 0$, we can construct a schedule ψ, such that $COST_{SA}(Q, \psi) > \gamma \cdot COST_{OPT}(Q, \psi) + \beta$. Thus, we have the following.

Proposition 1: For any constant γ and for any integer t, if $\gamma < 1 + c_c + c_d$, then the algorithm SA is not γ-competitive.

Proof : By way of contradiction, we suppose that there exist $\gamma < 1 + c_c + c_d$ and $\beta \geq 0$, such that $COST_{SA}(Q, \psi) \leq \gamma \cdot COST_{OPT}(Q, \psi) + \beta$ for any schedule ψ.

Consider the schedule ψ_1 that consists of m reads from j, $j \notin Q$, where m is big enough so that $m > \frac{\gamma \cdot (1 + c_c + c_d) + \beta}{1 + c_c + c_d - \gamma}$. Clearly, $COST_{OPT}(Q, \psi_1) \leq 1 + c_c + c_d + m$, and $COST_{SA}(Q, \psi_1) = (1 + c_c + c_d) \cdot m$. Then

$$\gamma \cdot COST_{OPT}(Q, \psi_1) + \beta \leq \gamma \cdot (1 + c_c + c_d + m) + \beta = \gamma \cdot (1 + c_c + c_d) + \beta + \gamma \cdot m$$

$$< (1 + c_c + c_d - \gamma) \cdot m + \gamma \cdot m = (1 + c_c + c_d) \cdot m = COST_{SA}(Q, \psi_1)$$

But this contradicts the hypothesis. □

4.2.2. The Dynamic Allocation Algorithm

Here we describe the DA algorithm presented in the introduction in terms of our model. DA selects a set of processors F of size $(t-1)$ and a processor $p \notin F$, such that $F \cup \{p\}$ is the initial allocation scheme. DA maps a schedule to an legal allocation schedule in the online steps as follows.

DA Algorithm

1. For a read request r^i, if i is a data processor, then DA associates with it the execution set $\{i\}$. Otherwise, i.e., if i is a non-data processor, then DA associates with it the execution set that consists of a processor in F, and DA turns this read into a saving read $\underline{r}^i$.
2. For a write request w^j, if $j \in F \cup \{p\}$, then DA associates with it the execution set $F \cup \{p\}$. Otherwise, i.e., if $j \notin F \cup \{p\}$, then DA associates with it the execution set $F \cup \{j\}$.

Theorem 2: For any integer t, the DA algorithm is $(2 + 2 \cdot c_c)$-competitive[8].

This theorem gives an upper bound of the competitiveness factor of DA, namely the bound $(2 + 2 \cdot c_c)$ may not be tight. We say so because we did not find a schedule for which the cost of DA is exactly $(2+2 \cdot c_c)$ $\times$ (the optimal cost) and we did not find a constant α ($\alpha < 2 + 2 \cdot c_c$) so that DA is α-competitive. Though we did not lower the competitiveness factor in general, we obtained that this factor can be lowered by c_c under the condition $c_d > 1$, i.e., the following.

Theorem 3: For any integer t, if $c_d > 1$, then the DA algorithm is $(2 + c_c)$-competitive[8].

Next, we obtain a lower bound of 1.5 for the competitiveness factor of DA, namely:

Proposition 2: For any constant γ and for any integer t, if $\gamma < 1.5$, then the algorithm DA is not γ-competitive.

Proof: By way of contradiction, we suppose that there exists a constant $\gamma < 1.5$ and a constant $\beta \geq 0$, such that $COST_{DA}(I, \psi) \leq \gamma \cdot COST_{OPT}(I, \psi) + \beta$ for any schedule ψ.

The initial allocation scheme is $I = F \cup \{p\}$ of size t. Let T be a set of processors of size t, such that $T \cap I = \emptyset$. Suppose that q is a processor, such that $q \notin I \cup T$. We denote by r^T the schedule that consists of t read requests issued from t different processors of T. Now consider the schedule $\psi_s = r^T w^q r^T w^p$, and

[8]The full proof of this theorem is given in the Appendix.

let ψ_2 be a schedule that is the concatenation of m subschedules ψ_s. Then,

$$COST_{DA}(I, \psi_2) = [(c_c + 2 + c_d) \cdot t + c_c \cdot (t+1) + (1 + c_d) \cdot t] \cdot 2 \cdot m$$

$$= 2 \cdot m \cdot [3 \cdot t + 2 \cdot c_d \cdot t + c_c \cdot (2 \cdot t + 1)]$$

where $(c_c + 2 + c_d) \cdot t$ is the cost of t read requests from T, and $c_c \cdot (t+1)$ is the control message cost for invalidating the $t+1$ copies (in $T \cup \{p\}$ or $T \cup \{q\}$) for the write (from q or p), and $(1 + c_d) \cdot t$ is the cost of writing to $F \cup \{q\}$ or to $F \cup \{p\}$ (depending on which is the writing processor).

Consider the algorithm B which does not have any saving-read and propagates every write to T. Then $COST_B(I, \psi_2) = (c_c + 1 + c_d) \cdot t + c_c \cdot t + (1 + c_d) \cdot t + [t + (1 + c_d) \cdot t] \cdot (2 \cdot m - 1)$. The OPT algorithm should cost at most as much as B costs for schedule ψ_2. Thus

$$COST_{OPT}(I, \psi_2) \leq (c_c + 1 + c_d) \cdot t + c_c \cdot t + (1 + c_d) \cdot t + [t + (1 + c_d) \cdot t] \cdot (2 \cdot m - 1)$$

$$= 2 \cdot t \cdot c_c - t + (1 + c_d) \cdot t + 2 \cdot m \cdot (2 \cdot t + t \cdot c_d)$$

Choose m to be big enough so that $m > \frac{3 \cdot t \cdot c_c + \beta + 1.5 \cdot (1 + c_d) \cdot t}{t \cdot c_d}$. Then,

$$\gamma \cdot COST_{OPT}(I, \psi_2) + \beta - COST_{DA}(I, \psi_2) \leq 1.5 \cdot COST_{OPT}(I, \psi_2) + \beta - COST_{DA}(I, \psi_2)$$

$$\leq 3 \cdot t \cdot c_c - 1.5 \cdot t + 2 \cdot m \cdot (3 \cdot t + 1.5 \cdot t \cdot c_d) + 1.5 \cdot (1 + c_d) \cdot t + \beta - 2 \cdot m \cdot (3 \cdot t + 2 \cdot t \cdot c_d)$$

$$< 3 \cdot t \cdot c_c + \beta + 1.5 \cdot (1 + c_d) \cdot t - m \cdot t \cdot c_d < 0$$

This contradicts the hypothesis. □

4.3 Static and Dynamic Allocation in Mobile Computing

Remember that in mobile computing with wireless communication the cost of servicing requests is dominated by the communication cost, i.e. the I/O cost is zero. We will first show that the static allocation algorithm, SA, is not competitive. Then we show that the dynamic allocation algorithm, DA, remains competitive.

Proposition 3: The SA algorithm is not competitive.

Proof: By way of contradiction, we suppose that there exist two constants $\gamma \geq 1$ and $\beta \geq 0$, such that $COST_{SA}(Q, \psi) \leq \gamma \cdot COST_{OPT}(Q, \psi) + \beta$ for any schedule ψ.

Consider the schedule ψ_1 that consists of m reads from j, where $j \notin Q$, and m is big enough so that $m > \gamma + \frac{\beta}{c_c + c_d}$. Clearly, $COST_{OPT}(Q, \psi_1) \leq COST_{DA}(Q, \psi_1) = c_c + c_d$. On the other hand, $COST_{SA}(Q, \psi_1) =$

$m \cdot (c_c + c_d) > \gamma \cdot (c_c + c_d) + \beta$, which contradicts the hypothesis. $\square$

Theorem 4: The DA algorithm is $(2 + 3 \cdot \frac{c_c}{c_d})$-competitive.

Proof: To prove this theorem, first we establish a lower bound on the communication cost of the optimal algorithm for an arbitrary schedule. Second, we establish an upper bound on the communication cost of the DA algorithm for the same schedule. Finally, we take the quotient of these two bounds to derive the competitiveness factor of the DA algorithm.

Assume that the initial allocation scheme $I = F \cup \{p\}$ is of size t. For an arbitrary given schedule with k writes, $k \geq 0$, we re-write the schedule as $\psi = \psi_0 \psi_1 \ldots \psi_k$, where ψ_0 is zero or more read requests which come before the first write, and ψ_i is the i^{th} write w^{p_i} followed by zero or more reads which come before the next write for $1 \leq i \leq k$. Denote by a_0 the number of different reading processors of ψ_0 which are not in $F \cup \{p\}$. For $1 \leq i \leq k$, denote by a_i the number of different reading processors of ψ_i which are not p_i. Denote by $\phi_0 \phi_1 \ldots \phi_k$ the OPT-allocated schedule of ψ, where ϕ_i corresponds to ψ_i. Denoted by J_i the allocation scheme at w^{p_i} using the OPT algorithm. Obviously from the definition of the $COST$ function, we conclude that

$$COST_{OPT}(I, \psi) = COST(I, \phi_0) + \sum_{i=1}^{k} COST(J_i, \phi_i) \qquad (*)$$

Denote by I_i the allocation scheme at w^{p_i} using the DA algorithm. Denote by ξ_i the sub-allocation schedule of the DA-allocated schedule of ψ so that ξ_i corresponds to ψ_i. Then we have

$$COST_{DA}(I, \psi) = COST(I, \xi_0) + \sum_{i=1}^{k} COST(I_i, \xi_i) \qquad (**)$$

Now we examine each item on the right hand side of the formula (**), and compare each piece with the corresponding estimated lower bound in (*).

To satisfy all the read requests in ψ_0, the communication cost is at least $a_0 \cdot (c_c + c_d)$, i.e., $COST(I, \phi_0) \geq a_0 \cdot (c_c + c_d)$. Observe that $COST(I, \xi_0) = a_0 \cdot (c_c + c_d)$ since every remote read is turned into a saving read in DA. Hence,

$$a_0 \cdot c_c \leq COST(I, \phi_0) \qquad (***)$$

Now consider the subschedules ψ_i for $i \geq 1$. The object written by p_i has to be transmitted to the a_i reading processors, hence the communication cost is at least $a_i \cdot c_d$. On the other hand, due to the t-available constraint, the object written by p_i has to be transmitted to $(t-1)$ other processors at w^{p_i}, which involves

a communication cost of $(t-1) \cdot c_d$. Therefore, we conclude that

$$COST(J_i, \phi_i) \geq max\{a_i \cdot c_d \ , \ \ (t-1) \cdot c_d\} \qquad (1)$$

Now we consider $COST(I_i, \xi_i)$. At the write, the 'invalidating' message cost is at most $(a_{i-1}+1) \cdot c_c$ since the allocation scheme I_i contains F and at most $(a_{i-1}+1)$ other processors, where c_c counts for the writing processor p_{i-1} or p, and $a_{i-1} \cdot c_c$ counts for the reading processors which join the allocation scheme after a read. The object transmission cost at the write is $(t-1) \cdot c_d$. All the reads in ξ_i cost at most $a_i \cdot (c_c + c_d)$. Therefore we have

$$COST(I_i, \xi_i) \leq (a_{i-1}+1) \cdot c_c + (t-1) \cdot c_d + a_i \cdot (c_c + c_d)$$

$$\leq_{(1)} (a_{i-1}+1) \cdot c_c + COST(J_i, \phi_i) + COST(J_i, \phi_i) \cdot (\frac{c_c}{c_d} + 1)$$

Thus, we obtain that for $i = 1, \ldots, k$,

$$COST(I_i, \xi_i) \leq (a_{i-1}+1) \cdot c_c + COST(J_i, \phi_i) \cdot (2 + \frac{c_c}{c_d}) \qquad (2)$$

By summing (2) up from 1 to k, we obtain

$$\sum_{i=1}^{k} COST(I_i, \xi_i) \leq \sum_{i=1}^{k} (a_{i-1}+1) \cdot c_c + \sum_{i=1}^{k} COST(J_i, \phi_i) \cdot (2 + \frac{c_c}{c_d})$$

$$\leq a_0 \cdot c_c + \sum_{i=1}^{k} (a_i + 1) \cdot c_c + \sum_{i=1}^{k} COST(J_i, \phi_i) \cdot (2 + \frac{c_c}{c_d})$$

Since $t \geq 2$, we derive $(a_i+1) \cdot c_c \leq a_i \cdot c_c + (t-1) \cdot c_c \leq_{(1)} COST(J_i, \phi_i) \cdot \frac{c_c}{c_d} + COST(J_i, \phi_i) \cdot \frac{c_c}{c_d} = 2 \cdot \frac{c_c}{c_d} \cdot COST(J_i, \phi_i)$. Therefore,

$$\sum_{i=1}^{k} COST(I_i, \xi_i) \leq a_0 \cdot c_c + (2 + 3 \cdot \frac{c_c}{c_d}) \cdot \sum_{i=1}^{k} COST(J_i, \phi_i) \qquad (3)$$

We derive,

$$COST_{DA}(I, \psi) =_{(**)} COST(I, \xi_0) + \sum_{i=1}^{k} COST(I_i, \xi_i)$$

$$\leq_{(3)} COST(I, \phi_0) + a_0 \cdot c_c + (2 + 3 \cdot \frac{c_c}{c_d}) \cdot \sum_{i=1}^{k} COST(J_i, \phi_i)$$

$$\leq_{(***)} (2 + 3 \cdot \frac{c_c}{c_d}) \cdot (COST(I, \phi_0) + \sum_{i=1}^{k} COST(J_i, \phi_i))$$

$$=_{(*)} (2 + 3 \cdot \frac{c_c}{c_d}) \cdot COST_{OPT}(I, \psi) \qquad \square$$

Observe that since $c_c \leq c_d$, the competitiveness factor of DA is at most 5.

5. Practical Issues and Extensions

In this section we concentrate on practical aspects of the DA algorithm (the implementation of static allocation has been studied extensively in the literature see [2, 3, 7, 27, 29, 30]). Particularly, we discuss the following issues. First, how the DA algorithm copes with failures. Second, the possibility that the read-write unit is different than the allocation unit. Third, extensions beyond the read-write model.

5.1 Failures and Recovery

In this subsection we consider transaction aborts and site failures. Transaction aborts are easy in the following sense. Assuming that an atomic commitment protocol is used for transactions in a distributed system, all the changes made by an aborting transaction, in particular the changes to the allocation scheme, are backed out.

Next we discuss site failures. At this stage of our research, we propose that dynamic allocation is used only during the normal mode of operation, i.e., when there are no site failures in the system. Thus our analysis holds for this case. When a site failure in F is detected, the system reverts to quorum consensus (see [17, 30]). The transition occurs using the missing writes algorithm (see [9, 3]). When a processor in F recovers from failure, the recovering procedure will bring the replicas in F up-to-date. Upon recovery of all processors in F, the system will resume to DA.

Now we elaborate upon the handling of failures. Suppose that the failures are clean (see [15]), i.e. they can be detected and a failed processor is totally down. A processor failure may occur at a data processor (a processor that has a replica of the object) or a non-data processor. The failure of a non data processor does not affect the execution of read-write requests at the other processors. These failures can be simply ignored.

In DA each processor q ($q \notin F$) has a unique "server", namely a processor of F that receives the read/write request of q and invalidates the copy at q. Suppose that a data processor d fails, and $d \notin F$. Denote its server by s_d. Then, d must be in the "join-list" of s_d. When the failure of d is detected, s_d will remove d from its "join-list", and continue. Upon recovery, processor d will consider itself as a non-data processor.

Suppose now that a processor s in F fails. When a processor q that attempts to read from s detects the failure of s, it simply re-directs the read to another processor of F. When a writing transaction detects a 'missing write' to s, then the system resorts to the 'Quorum Consensus' protocol with the static allocation scheme that consists of the processors in F. Upon recovery of s, it tries to x-lock all replicas in F. If unsuccessful, namely some other processor in F is still down, then s will read from a majority of F and

bring its own copy up-to-date. Otherwise, i.e., if all processors in F are 'up', then DA is resumed as follows. s informs all processors of F that there is no missing write. A non-server processor that attempts to read from a quorum is informed that there is no missing write, and is instructed to direct its request to its own server.

Further details of failure handling and proof of correctness are beyond the scope of this paper.

5.2 Discrepency between the read-write unit and the allocation/replication unit

The model discussed in this paper assumes that the read-write unit is identical to the replication unit. In other words, an object was both, the unit being allocated and replicated, as well as the unit being read and written. This is often the case when data is replicated in logical units (e.g. a text file) but not when data is allocated in physical units. For example, suppose that the replication unit is a block, i.e., data is replicated in full blocks. Furthermore, suppose that as a result of a read-write request, the unit transferred between two processors is a set of tuples out of a block. Then, as a result of servicing a read-write request, only a fraction of the replication unit is being transferred. Our model can be adapted to handle this situation by representing a schedule as a sequence of requests, as before, except that with each one we associate a fraction; it is the fraction of the physical object that is being requested. Obviously, we can obtain a lower bound for this extended-model as well.

The dynamic allocation DOM can be adapted for this case by, for example, imposing the following rule. The object o is replicated at processor p, when the total size of transfers to p (as a result of reads issued by p) exceeds the size of the last write of o. We conjecture that the DA algorithm modified in this fashion remains competitive.

5.3 Method invocation in object oriented environments

We can also extend our model beyond reads and writes. In object-oriented systems processors do not issue read write requests, but invoke methods. However, a method invocations also either results in the modification of the object, or it results in some information from the object being processed and the result transmitted. When a method that modifies an object is being invoked, the invocation can be modeled as a write request with two parameters. One is the amount of data the caller-processor transmits to the callee-processor as parameters for the method, and the other is the amount of data that the callee has to output to secondary storage to modify the object. Therefore, in an object-oriented system a schedule is a sequence of requests, each of which has two associated parameters: the I/O cost of servicing the request,

and the communication cost of servicing the request. A lower bound can also be obtained for this model, and dynamic allocation algorithms can be devised, but this is the subject of future research.

6. Comparison with Relevant Literature

One research area that is relevant to our dynamic allocation approach is caching in various contexts, e.g. networking and the World-Wide Web (see for example [21, 26, 13, 14, 6, 5, 16]), Client/Server databases (e.g. [34, 11]), distributed file systems (e.g. [23, 20, 28, 12]), and distributed shared memory(e.g. [22, 25]). However, our approach has several important aspects whose combination is unique. First, we study dynamic allocation as an independent concept, unrestricted by the limitations of a particular system, protocol, or application. In existing studies, a particular environment often restricts the allocation options. Second, we consider caching in a peer-to-peer rather than client-server environment. Third, we consider the dollar cost in addition to performance. Fourth, we perform a formal worst-case analysis. As far as we know, these aspects have not been studied in combination in any of the existing works.

In [32, 19, 18] we considered the cost and time of dynamic allocation. However, the model there ignores the I/O cost and availability constraints (i.e., they consider only communication). Also, the model in [32] is dependent on the communication network having a tree topology. The present paper removes this dependency.

Additionally, the algorithms developed in [32] are convergent rather than competitive. This means the following. Assume that the pattern of access to each object is generally regular. For example, during the first two hours processor x executes three reads and one write per second, processor y executes five reads and two writes per second, etc.; during the next four hour period processor x executes one read and one write per second, processor y executes two reads and two writes, and so on. Then, a convergent algorithm will move to the optimal allocation scheme for the global read-write pattern during the first two hours, then it will converge to the optimal allocation scheme for the global read-write pattern during the next four hours, etc.

Competitiveness and convergence are two criteria for evaluating online algorithms. A competitive algorithm may not converge to the optimal allocation scheme when the read-write pattern stabilizes, and a convergent algorithm may unboundedly diverge from the optimum when the read-write pattern is irregular. A competitive online algorithm is more appropriate for chaotic read-write patterns, in which the past access pattern does not provide any indication to the future read-write pattern. In contrast, a convergent

online algorithm is more appropriate for regular read-write patterns (although it is not guaranteed that a convergent algorithm will always outperform a competitive one, even for regular read-write patterns).

In [33] we developed a different algorithm that is also convergent rather than competitive. In [19] we developed a competitive algorithm, called CDDR, for a model that ignores the I/O cost and the availability constraints. The DA algorithm presented here is totally different than CDDR. Furthermore, the CDDR algorithm is not competitive when the I/O cost and the availability constraints are taken into consideration.

There has also been work addressing dynamic object allocation algorithms in [1]. However, the model there does not allow concurrent requests, and it requires centralized decision making by a processor that is aware of all the requests in the network. In contrast, our algorithms are distributed, and allow concurrent read-write requests. Additionally, the model in [1] also concentrates exclusively on communication, and it does not consider I/O cost and availability constraints.

The two main purposes of replicated data are increased availability and performance. No other work of which we are aware, quantitatively compares static and dynamic allocation, while considering both purposes of data replication and treating I/O and communication costs in a unified model.

Static allocation was studied in [31, 8], however not from an online-algorithmic point of view. In other words, these works address the following file-allocation problem. They assume that the read-write pattern at each processor is known a priori and they find the optimal static allocation scheme. However, works on file-allocation problem do not quantify the cost penalty if the read-write pattern is not known. In contrast, in this paper we do so.

7. Conclusion

In this paper we addressed the problem of automatic allocation and replication of a single object in a distributed system. Particularly, we considered distributed-object-management (DOM) algorithms. Such an algorithm services read-write requests by mapping the request to a set of processors that will execute it. Additionally, the algorithm determines the allocation scheme of the object, namely the set of processors that store the object in their local database. The allocation scheme may change over time, as read-write requests are processed.

We introduced a model for analyzing the communication cost of distributed object management algorithms, and a model for analyzing both the communication cost and the I/O cost of such algorithms in an integrated fashion. We discussed distributed-object-management algorithms on two dimensions. First, the knowledge level of the algorithm. Offline algorithms have a priori knowledge of all read-write requests,

whereas online algorithms do not do so; they have to service a request and determine the allocation scheme without knowing the future requests. Offline algorithms were used as a yardstick to evaluate the performance of online algorithms. The second dimension is variation of the allocation scheme. A static algorithm does not vary the allocation scheme while processing reads and writes, whereas a dynamic algorithm does so.

We also introduced the dynamic allocation (DA) algorithm. In DA, whenever a processor reads a copy of the object, it saves the copy in its local database. Then we compared the DA algorithm with the traditional, read-one-write-all, static allocation (SA) algorithm. We showed that in the model where both communication and I/O costs are considered, the SA algorithm is tightly $(1+c_c+c_d)$-competitive, the DA algorithm is $(2+2 \cdot c_c)$-competitive, and when $c_d > 1$ the DA algorithm is $(2+c_c)$-competitive; c_c is the ratio of the cost of transmitting a control message to the cost of inputting/outputting the object to the local database on secondary storage, and c_d is the ratio of the cost of transmitting the object between two processors to the I/O cost. A DOM algorithm is α-competitive if the ratio of the cost of the algorithm to the optimal cost is at most α, for an arbitrary sequence of read-write requests. Also, we showed that in the model where only communication cost matters (e.g. mobile computing), the SA algorithm is not competitive while the DA algorithm remains competitive. Hence in this model the DA algorithm is strictly superior to the SA algorithm as demonstrated in figure 2 of the introduction.

A point that we would like to emphasize here is that the DA algorithm is not simply the SA algorithm, modified to save in the local database every object that is read remotely. Rather, in DA, the set of t processors to which a write is propagated changes (although the set always contains F as a subset) depending on the writing processor. We also considered another algorithm, called Expanding-Allocation(EA) (EA is not discussed in the paper). In EA, every write is propagated to a fixed set of t processors (as in SA), and every remote read saves a copy of the object in the local database (as in DA). In EA, the competitiveness factor is higher than in DA.

In the stationary computing model, where both communication and I/O costs are considered, the results of the comparison between the DA and SA algorithms are that SA is superior when $c_c + c_d < 0.5$, and DA is superior when $c_d > 1$. These results are summarized in figure 1 of the introduction. The intuition for these comparison results is as follows. At some level of abstraction, the DA algorithm is similar to the SA algorithm, except that in DA a processor that reads the object from another one joins the allocation scheme, whereas in SA it does not do so. By joining the allocation scheme a processor incurs the following tradeoff. The current read becomes more expensive by 1, namely the I/O cost, and future reads that occur before

the next write are less expensive by $c_c + c_d$, namely the communication cost of requesting and receiving an object. Therefore SA is superior when the communication costs are small compared to the I/O cost, and DA is superior when the opposite occurs. So why isn't it the case that simply SA is superior when $c_c + c_d < 1$, and DA is superior when $c_c + c_d > 1$? The answer to this question is that our objective function is global as opposed to local, and in the global sense the cost of saving a copy of the object in the local database is not only the I/O cost. Each subsequent write will also have to be propagated to a processor that joins the allocation scheme, and this costs communication. So, a processor p that keeps a copy of the object not only pays the I/O cost of saving the object in the local database, but will also have to pay the price of having updates of the object propagated to p.

The area of Figure 1 that is marked "Unknown" represents the c_c and c_d values for which it is currently unknown whether the DA algorithm is superior to the SA algorithm or vice versa. The reason for this uncertainty is that there is a gap between the upper and lower bound on the competitiveness of the DA algorithm. This gap is the subject of future research.

References

[1] Y. Bartal, A. Fiat, Y. Rabani, "Competitive Algorithms for Distributed Data Management", *24th Annual ACM STOC*, 5/92, Victoria, B.C. Canada.

[2] P.A. Bernstein, and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases", *ACM-TODS*, Vol. 9, No. 4, December 1984, Pages 596-615

[3] P. Bernstein, V. Hadzilacos, and N. Goodman, "Concurrency control and recovery in database systems", *Addison-Wesley* (1987)

[4] David L. Black and Daniel D. Sleator, "Competitive Algorithms for Replication and Migration Problems", *Technical Report*, CMU-CS-89-201, November 1989

[5] M. E. Crovella, R. L. Carter, "Dynamic Server Selection in the Internet", *Proc. of the 3rd Workshop on the Architecture and Implementation of High Performance Communication Subsystems, 95.*

[6] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz,, K. J. Worrell, "A Hierarchical Internet Object Cache", *Proc. of USENIX, 1996.*

[7] Michael J. Carey, and Miron Livny, "Distributed concurrency control performance: A study of algorithms, distribution and replication", *Proc. of the 14th VLDB Conf.*, Los Angeles, CA, 1988

[8] L. W. Dowdy and D. V. Foster, "Comparative Models of the File Assignment Problem', *ACM Computing Surveys*, 14 (2), 1982.

[9] Eager, D.L. and K.C. Sevick, "Achieving Robustness in Distributed Database Systems", *ACM Trans. on Database Systems*, 8(3), 354-381, Sept. 1983

[10] A. Fiat, R, Karp, M.Luby, L.A. McGeoch, D.Sleator, N.E. Yong, "Competitive paging algorithms", *Journal of Algorithms*, 12, pages 685-699, 1991

[11] M.J. Franklin, "Client Data Caching", *Kluwer Academic Publishers, 1996.*

[12] C. Gray, D. Cheriton, "Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency", *Proceedings of the 12th International Symposium on Operating System Principles*, 1989.

[13] J. Gwertzman, M. Seltzer, "World-Wide Web Cache Consistency", *Proceedings of the 1996 Usenix Conference*, January 1996.

[14] J. Gwertzman, M. Seltzer, "The Case for Geographical Push Caching", *Workshop on Hot Operating Systems*, 1995.

[15] N. Goodman, D. Skeen, A. Chan, U.Dayal, S. Fox, D. Ries, "A recovery algorithm for a distributed database system", *Proc. 2nd ACM SIGACT-SIGMOD, Symp. Database System*, Atlanta, GA, March 1983, pp 8-15

[16] J. D. Guyton, M. F. Schwartz, "Locating Nearby Copies of Replicated Internet Servers", TR CU-CS-762-95 Department of Computer Science, Univ. of Colorado, Boulder, February 1995.

[17] D. Gifford. "Weighted Voting for Replicated Data", *Proc. of 7th ACM Symposium on Operation System Principles*, pages 150-162, 1979

[18] Yixiu Huang, Prasad Sistla and Ouri Wolfson, "Data Replication for Mobile Computers", *ACM-SIGMOD International Conference on Management of Data*, May 1994, Minneapolis, MN.

[19] Yixiu Huang, Ouri Wolfson, "A Competitive Dynamic Data Replication Algorithm", *IEEE Proc. of 9th Int'l Conf. on Data Engineering* '93, pp. 310-317, Vienna, Austria

[20] J. Kistler, M. Satyanarayanan, "Disconnected Operation inn the Coda File System", *Proceedings of the 13th International Symposium on Operating System Principles*, Pacific Grove, CA, Oct. 1991.

[21] C. Liu, P. Cao, "Maintaining Strong Cache Consistency in the World-Wide Web", *Proc. of the International Conference on Distributed Computing Systems, 97.*

[22] K. Li, P. Hudak, "Memory Coherence in Shared Virtual Memory Systems", *ACM Transactions on Computer Surveys*, Vol 7, No. 4, Nov. 1989.

[23] E. Levy, A. Silbershatz, "Distributed File Systems: Concepts and Examples", *ACM Computing Surveys*, Vol. 22, No. 4, Dec. 1990.

[24] M. Manasse, L.A. McGeoch, and D.Sleator, "Competitive algorithms for online problems", *Proc. 20th ACM STOC*, page 322-333, ACM 1988

[25] B. Nitzberg, V. Lo, "Distributed Shared Memory: A Survey of Issues and Algorithms", *IEEE Computer*, Vol. 24, No. 8, August 1991.

[26] K. Obraczka, P. Danzig, D. DeLucia, E-Y. Tsai, "A Tool Massively Replicating Internet Archives: Design, Implementation, and Experience", *Proc. of the International Conf. on Distributed Computing Systems, 1996.*

[27] T. M. Ozsu, P. Valduriez, "Principles of Distributed Database Systems", *Prentice Hall*

[28] G. Popek, R. Guy, T. Page, J. Heidemann, "Replication in Ficus Distributed File Systems", *Proceedings of the Workshop on Management of Replicated Data*, IEEE, Houston, Nov. 1990.

[29] C. Papadimitriou, "The serializability of concurrent updates", *Journal of the ACM*, 26(4), pp. 631-653

[30] R. H. Thomas, "A Majority Consensus Approach to Concurrency Control for Multiple Copy Database", *ACM Trans. on Database Systems*, 4(2):180-209, June, 1979

[31] O. Wolfson and A. Milo, "The Multicast Policy and Its Relationship to Replicated Data Placement", *ACM TODS*, 16 (1), 1991.

[32] Ouri Wolfson and Sushil Jajodia, "Distributed Algorithms for Dynamic Replication of Data", *Proc. of ACM-PODS*, 1992

[33] Ouri Wolfson and Sushil Jajodia, "An Algorithm for Dynamic Data Distribution", *Proc. of the 2nd Workshop on Management of Replicated Data (WMRD-II)*, 1992, pp. 62-65

[34] M. Zahariousdakis, M. J. Carey, "Hierarchical, Adaptive Cache Consistency in a Page Server OODBMS", *Proc. of the International Conference on Distributed Computing Systems, 97.*

APPENDIX : Competitiveness Proof of the DA Algorithm

In this appendix, we prove theorem 2 and theorem 3.

A.1 Outline of the Proof

Given an initial allocation scheme $I = F \cup \{p\}$ of size t, we will prove that for any schedule ψ, $COST_{DA}(I, \psi) \leq \alpha \cdot COST_{OPT}(I, \psi)$. In general, $\alpha = 2 + 2 \cdot c_c$, and if $c_d > 1$ then $\alpha = 2 + c_c$. The proof proceeds in the following steps.

First, we decompose the objective function $COST$ into two functions, namely, CO and ST. For any DOM algorithm A, the first function $CO_A(I, \psi)$ denotes the total control message cost for processing the schedule ψ using algorithm A. It is a multiple of c_c (refer to the definition of $COST$ in section 2.2). The second function $ST_A(I, \psi)$ denotes the total other cost of processing the schedule, namely the total cost incurred for inputting/outputting the object from/to the local databases and the data message cost. Obviously, $COST$ is the sum of the CO-cost and the ST-cost, i.e.,

$$COST_A(I, \psi) = CO_A(I, \psi) + ST_A(I, \psi) \qquad (1.1)$$

Second, we devise an offline t-LB algorithm, and show that for arbitrary schedule the t-LB-allocated schedule reaches the lower bound for the ST function, i.e., $ST_{t\text{-}LB}(I, \psi) \leq ST_A(I, \psi)$ for any DOM algorithm A. Thus,

$$ST_{t\text{-}LB}(I, \psi) \leq ST_{OPT}(I, \psi) \leq COST_{OPT}(I, \psi) \qquad (1.2)$$

Notice that if we ignore the control message cost, i.e., we assume $c_c = 0$, then t-LB is the optimum algorithm.

Third, we will show that $ST_{DA}(I, \psi) \leq 2 \cdot ST_{t\text{-}LB}(I, \psi)$, and $CO_{DA}(I, \psi) \leq 2 \cdot c_c \cdot ST_{t\text{-}LB}(I, \psi)$. Additionally, we show that if $c_d > 1$ then $CO_{DA}(I, \psi) \leq c_c \cdot ST_{t\text{-}LB}(I, \psi)$. These three steps combined prove the theorem 2 and 3.

The rest of this appendix is organized as follows. In A.2, we describe the t-LB Algorithm. In A.3, we prove that t-LB-allocated schedule has the lower bound for the ST function. Lastly in A.4, we prove the theorem 2 and the theorem 3.

A.2 The t-LB Algorithm

In this section we present a DOM algorithm called t-LB. t-LB is defined for an integer t, and it is t-available constrained. It is designed to minimize the ST-cost of a schedule, while ignoring the CO-cost.

The t-LB algorithm maps a schedule ψ to a corresponding legal allocation schedule $las_{t\text{-}LB}(\psi)$; it does so by sequentially examining each request in ψ, starting with the first one. Intuitively, t-LB computes a tradeoff function δ for each request. Based on δ, t-LB determines what execution set to associate with the request and which read to turn into a saving-read. The function δ at a request q depends on requests that succeed q, and therefore t-LB is an off-line algorithm. A read request from a non-data processor i is turned into a saving read if the extra cost for outputting the object to the local database is paid off by eliminating the communication cost of future reads from i. With a write request, t-LB associates an execution set which, basically, consists of the t processors which issue the most reads between the current write and the next one. However, in choosing these t processors the writing processor receives a special treatment, because writing locally does not involve communication cost.

The t-LB Algorithm

1. For r^i, i.e., a read request from processor i,
   (a) If i is a data processor at r^i, then t-LB associates with it the set $\{i\}$.
   (b) If i is a non-data processor at r^i, then t-LB associates with it the set that consists of an arbitrary single processor of the allocation scheme. Additionally, t-LB computes $\delta(i) = n_i \cdot c_d - 1$, where n_i is the number of reads that i issues before the next write, but after r^i. If $\delta(i) > 0$, then t-LB turns this read into a saving-read.
2. For w^j, i.e., a write request from processor j, t-LB computes $\delta(j) = n_j \cdot c_d - 1$, and $\delta(i) = n_i \cdot c_d - c_d - 1$ for every other processor i in the network; n_j (n_i) is the number of reads that processor j (i) issues between w^j and the next write-request.
   (a) If $\delta(j) > 0$, then t-LB associates with w^j the execution set that consists of j and $(t-1)$ other processors that have the largest δ-values.
   (b) Otherwise, i.e., if $\delta(j) \leq 0$, then t-LB associates with w^j the set of t processors that have the largest δ-values.

The following example demonstrates the t-LB algorithm.

Example 1: Suppose that the network consists of four processors $\{1,2,3,4\}$, and the initial allocation scheme is $\{1,2\}$. Suppose that the relative communication cost $c_d = 1.5$, and the reliability threshold $t = 2$. Consider the request schedule $\psi_1 = r^3r^4r^3r^1r^3w^2r^3r^2r^4r^3r^4r^1r^3$. t-LB associates with the first read (r^3) the execution set $\{1\}$. Since processor 3 will issue two more reads before the next write, and since $\delta(3) = 2 \cdot 1.5 - 1 > 0$, t-LB turns this read into a saving-read. t-LB associates with the second read (r^4), the execution set $\{1\}$. Because processor 4 issues no more reads before the next write, the copy is not saved in the local database at processor 4. The third, fourth, and fifth requests are issued from data processors, hence the execution sets consist of the reading processor alone. For the sixth request (w^2), t-LB computes the δ-function for every processor. The number of reads issued before the next write (actually there is no next write) by the various processors are: $n_1 = n_2 = 1, n_3 = 3, n_4 = 2$. Hence, $\delta(1) = -1, \delta(2) = 0.5, \delta(3) = 2$, and $\delta(4) = 0.5$. Therefore, t-LB associates with this write the execution set $\{2,3\}$. For the seventh, and eighth requests (reads from data processors), t-LB associates with each one of them the sets consisting of the reading processor alone. t-LB associates with the ninth request (r^4), the execution set $\{2\}$. Because processor 4 issues one more read afterwards, and $\delta(4) = 1 \cdot 1.5 - 1 > 0$, t-LB turns this read into a saving-read. The tenth, eleventh, and thirteenth requests are reads from data processors, thus t-LB associates with them the sets consisting of the reading processor alone. t-LB associates with the twelfth request (r^1), the execution set $\{2\}$. Because processor 1 issues no more read afterwards, the copy is not saved at processor 1. Therefore the t-LB-allocated schedule of ψ_1 is $las_{t\text{-}LB}(\psi_1) =$

$$\underline{r}^3\{1\}, r^4\{1\}, r^3\{3\}, r^1\{1\}, r^3\{3\}, w^2\{2,3\}, r^3\{3\}, r^2\{2\}, \underline{r}^4\{2\}, r^3\{3\}, r^4\{4\}, r^1\{2\}, r^3\{3\} \quad \square$$

We define a t-available constrained DOM algorithm B to be *ST-optimal* if for an arbitrary initial allocation scheme I and an arbitrary schedule ψ, the inequality $ST_B(I,\psi) \leq ST_A(I,\psi)$ holds true for any t-available constrained DOM algorithm A.

Theorem 5: For any integer t, the t-LB algorithm is ST-optimal.

A.3 Proof of the Theorem 5

A.3.1 Proof Outline

First, we study two special types of schedule, namely 1) the schedule which consists of read requests only; and 2) the schedule which consists of a write followed by zero or more read requests. And we show that t-LB has the minimum ST-cost on these two types of schedules among all t-available constrained DOM algorithms.

Second, we investigate the ST-cost of a DOM algorithm on an arbitrary given input schedule ψ. We split the schedule into several sub-schedules, each one of which falls into one of the two special types of schedule mentioned above. And we show that the ST-cost of a DOM algorithm on the whole schedule is the sum of the ST-costs of the DOM algorithm on each one of these sub-schedules.

At last, we use the results to conclude the theorem 5.

The technique that we use for the proof is as follows. To examine the total ST-cost of an allocation schedule, we re-formulate the ST-cost of each request in terms of the sum of p-costs (processor-cost) at different processors. Intuitively, the p-cost of a processor q at a request o is defined as follows. If o is a read issued by q, then the p-cost of q is the total ST-cost of the request and the p-cost of other processors are zeros. If o is a write issued by y, then the p-cost of q is the ST-cost incurred at q. Namely, if q is not in the execution set of o, then the p-cost at q is zero; If q is in the execution set of o and $q = y$, then the p-cost at q is 1; Otherwise, i.e., if q is in the execution set of o and $q \neq y$, then the p-cost at q is $1 + c_d$. The p-cost of a processor on a schedule is the sum of the p-costs of the processor at all the requests of the schedule. After defining the p-cost, we analyze the t-LB-allocated schedule and A-allocated schedule of the same arbitrarily given schedule by comparing their p-cost on the whole schedule at each processor in order to conclude the theorem.

The rest of section A.3 consists of the following subsections. In A.3.2, we describe the ST-cost re-formulation in detail and formally define the p-cost. In A.3.3, we show that t-LB has the minimum ST-cost on the first type of schedules. In A.3.4, we show that t-LB has the minimum ST-cost on the second type of schedules. In A.3.5, we show that the ST-cost of an arbitrary DOM algorithm A on an arbitrary schedule is the sum of the ST-cost of the algorithm A on several subschedules. Lastly in A.3.6, we prove the theorem 5.

A.3.2 Re-formulating the ST-cost

Given an initial allocation scheme I, and an input schedule ψ, for the DOM algorithm A-allocated schedule $las_A(\psi) = o_1^{j_1} X_1 o_2^{j_2} X_2 \ldots o_m^{j_m} X_m$, we define the *p-cost* of processor q at the request $o_i^{j_i}$, denoted by

$PC_A^q(o_i^{j_i})$, as follows.

$$PC_A^q(o_i^{j_i}) = \begin{cases} 0 & \text{if } o_i^{j_i} \text{ is a read and } q \neq j_i \\ ST_A(I, o_i^{j_i}) & \text{if } o_i^{j_i} \text{ is a read and } q = j_i \\ 0 & \text{if } o_i^{j_i} \text{ is a write and } q \notin X_i \\ 1 & \text{if } o_i^{j_i} \text{ is a write, } q = j_i, \text{ and } q \in X_i \\ 1 + c_d & \text{if } o_i^{j_i} \text{ is a write, } q \neq j_i, \text{ and } q \in X_i \end{cases}$$

¿From this definition we see that $ST_A(I, o_i^{j_i}) = \sum_{q=1}^{N} PC_A^q(o_i^{j_i})$, where N is the number of processors in the system.

We define the *p-cost* of processor q on ψ to be the sum of the p-costs of q at all requests in ψ, namely, $PC_A^q(\psi) = \sum_{i=1}^{m} PC_A^q(o_i^{j_i})$. Then

$$ST_A(I, \psi) = \sum_{i=1}^{m} ST_A(I, o_i^{j_i}) = \sum_{i=1}^{m}\sum_{q=1}^{N} PC_A^q(o_i^{j_i}) = \sum_{q=1}^{N}\sum_{i=1}^{m} PC_A^q(o_i^{j_i}) = \sum_{q=1}^{N} PC_A^q(\psi)$$

A.3.3 For Read-Only Schedules – the First Type of Schedules

Lemma 3.1: If I is an arbitrary given initial allocation scheme, and π is a schedule that consists of read requests only, then $ST_{t\text{-}LB}(I, \pi) \leq ST_A(I, \pi)$ for any t-available constrained DOM algorithm A.

Proof: Suppose $\pi = r^{j_1} r^{j_2} \ldots r^{j_k}$. Denote by X the set of reading processors in π. Denote by m_j the number of reads from j in π.

Case i: $\underline{j \notin X}$, $PC_{t\text{-}LB}^j(\pi) = PC_A^j(\pi) = 0$.

Case ii: $\underline{j \in X \cap I}$, $PC_{t\text{-}LB}^j(\pi) = PC_A^j(\pi) = m_j$;

Case iii: $\underline{j \in X/I, \text{ and } (m_j - 1) \cdot c_d - 1 > 0}$. Then in π, the first time processor j reads, we have $n_j = m_j - 1$ and $\delta_j > 0$. Hence t-LB turns this first read into a saving-read and $PC_{t\text{-}LB}^j(\pi) = 1 + c_d + m_j$.

- **a:** If A turns j's n^{th} read in into a saving-read, where $1 \leq n \leq m_j$. Then $PC_A^j(\pi) = n \cdot (1 + c_d) + 1 + m_j - n = m_j + n \cdot c_d + 1 \geq PC_{t\text{-}LB}^j(\pi)$.
- **b:** If A does not turn j's read into a saving-read at all, then $PC_A^j(\pi) = (1 + c_d) \cdot m_j = m_j + c_d \cdot m_j > m_j + c_d + 1 = PC_{t\text{-}LB}^j(\pi)$.

Case iv: $\underline{j \in X/I, \text{ and } (m_j - 1) \cdot c_d - 1 \leq 0}$. In π processor j always have $\delta_j \leq 0$, hence t-LB does not turn j's read into a saving-read at all, and $PC_{t\text{-}LB}^j(\pi) = (1 + c_d) \cdot m_j$. If A turns j's n^{th} read in into a

saving-read, where $1 \leq n \leq m_j$, then $PC_A^j(\pi) = n \cdot (1 + c_d) + 1 + m_j - n = m_j + (n-1) \cdot c_d + c_d + 1 \geq m_j + m_j \cdot c_d = PC_{t\text{-}LB}^j(\pi)$.

Summarizing above cases, we see that for every processor j, $PC_{t\text{-}LB}^j(\pi) \leq PC_A^j(\pi)$. Therefore

$$ST_{t\text{-}LB}(I, \pi) = \sum_{j=1}^{N} PC_{t\text{-}LB}^j(\pi) \leq \sum_{j=1}^{N} PC_A^j(\pi) = ST_A(I, \pi) \qquad \square$$

A.3.4 For the Second Type of Schedules

Lemma 3.2: If I is an arbitrary given initial allocation scheme, and σ is a schedule that consists of a write followed by zero or more read requests, then $ST_{t\text{-}LB}(I, \sigma) \leq ST_A(I, \sigma)$ for any t-available constrained DOM algorithm A.

Proof: Suppose $\sigma = w^{j_0} r^{j_1} \ldots r^{j_k}$. Denote by T the execution set of the write w^{j_0} using algorithm t-LB. Then $|T| = t$. Denote the set of processors which satisfy $\delta(j) > 0$ at the write by Y, and $y = |Y|$. Now we analyze the case i) $y \geq t$, and the case ii) $y < t$ respectively. We denote by m_j the number of reads of processor j in σ. Then $n_j = m_j$ at the write for all j.

Case i: $\underline{y \geq t}$. In this case we investigate the p-cost at j_0, the p-cost of a processor in Y, and the p-cost of a processor not in Y, respectively.

a: $\underline{j_0 \in Y}$. Then $j_0 \in T$ and $PC_{t\text{-}LB}^{j_0}(\sigma) = 1 + m_{j_0}$. $\delta(j_0) = c_d \cdot m_{j_0} - 1 > 0$.

1. If A turns j_0's n^{th} read into a saving read, where $0 \leq n \leq m_{j_i}$, $n = 0$ implies that processor j_0 gets a copy when at the write. Then $PC_A^{j_0}(\sigma) \geq (1 + c_d) \cdot n + 1 + m_{j_i} - n = m_{j_i} + n \cdot c_d + 1 \geq PC_{t\text{-}LB}^{j_0}(\sigma)$.
2. If A does not output the object to j_0's local database at all, then $PC_A^{j_0}(\sigma) \geq (1 + c_d) \cdot m_{j_0} = m_{j_0} + c_d \cdot m_{j_0} > m_{j_0} + 1 = PC_{t\text{-}LB}^{j_0}(\sigma)$.

b: $\underline{j_0 \notin Y}$. Then $j_0 \notin T$, $PC_{t\text{-}LB}^{j_0}(\sigma) = (1 + c_d) \cdot m_{j_0}$ and $\delta(j_0) = c_d \cdot m_{j_0} - 1 \leq 0$. If A turns j_0's n^{th} read into a saving-read, where $0 \leq n \leq m_{j_i}$. Then $PC_A^{j_0}(\sigma) \geq (1 + c_d) \cdot n + 1 + m_{j_0} - n = m_{j_i} + n \cdot c_d + 1 \geq m_{j_i} + n \cdot c_d + c_d \cdot m_{j_i} \geq PC_{t\text{-}LB}^{j_0}(\sigma)$.

c: $\underline{j \neq j_0 \text{ and } j \in Y}$. Then $\delta(j) = c_d \cdot (m_j - 1) - 1 > 0$. t-LB either puts a replica at j at the write or turns j's first read into a saving-read, therefore $PC_{t\text{-}LB}^j(\sigma) = 1 + c_d + m_j$.

1. If A turns j's n^{th} read into a saving read, where $1 \leq n \leq m_j$, then $PC_A^j(\sigma) \geq (1 + c_d) \cdot n + 1 + m_j - n = m_j + n \cdot c_d + 1 \geq PC_{t\text{-}LB}^j(\sigma)$

2. If A does not output the object to j's local database at all, then $PC_A^j(\sigma) = (1 + c_d) \cdot m_j = m_j + c_d \cdot m_j \geq PC_{t\text{-}LB}^j(\sigma)$

d: $\underline{j \neq j_0 \text{ and } j \notin Y}$. Then $\delta(j) = c_d \cdot (m_j - 1) - 1 \leq 0$. t-LB does not put a replica at j's local database, hence $PC_{t\text{-}LB}^j(\sigma) = (1 + c_d) \cdot m_j$.

1. If A puts a replica to j's local database at the write, then $PC_A^j(\sigma) = 1 + c_d + m_j \geq c_d \cdot m_j + m_j = PC_{t\text{-}LB}^j(\sigma)$.
2. If A turns j's n^{th} read into a saving read, where $1 \leq n \leq m_j$, then $PC_A^j(\sigma) = (1 + c_d) \cdot n + 1 + m_j - n = m_j + n \cdot c_d + 1 \geq m_j + c_d + 1 \geq PC_{t\text{-}LB}^j(\sigma)$.

In this case we see that for every processor j, $PC_{t\text{-}LB}^j(\sigma) \leq PC_A^j(\sigma)$. Therefore we obtain $\sum_{j=1}^N PC_{t\text{-}LB}^j(\sigma) \leq \sum_{j=1}^N PC_A^j(\sigma)$

Case ii: $\underline{y < t}$. In this case, t-LB turns no read into a saving-read. Denote by X the execution set of w^{j_0} using A. Then $|X| \geq t$. Let $R = T/X$, and $r = |R|$. Then there must be r different processors in X/T. Suppose that $R = \{i_1, \ldots, i_r\}$, and $K = \{k_1, \ldots, k_r\} \subset X/T$. We will compare the p-cost at processor j for the following cases. a) $j \in X \cap T$; b) $j \notin T \cup K$; c) $j \in R \cup K$.

a: $\underline{j \in X \cap T}$.

$$PC_A^j(\sigma) \geq PC_{t\text{-}LB}^j(\sigma) = \begin{cases} 1 + m_j & \text{if } j = j_0 \\ 1 + c_d + m_j & \text{otherwise} \end{cases}$$

b: $\underline{j \notin T \cup K}$. $PC_{t\text{-}LB}^j(\sigma) = (1 + c_d) \cdot m_j$. Algorithm A may put a replica at j at the write, or turn the n^{th} read of j into a saving-read for $1 \leq n \leq m_j$, or do not put a copy at j at all. We investigate these cases respectively.

1. $\underline{j \in X, \text{ and } j = j_0}$. Then we know from the *t-LB* algorithm that $c_d \cdot m_{j_i} \leq 1$. $PC_A^j(\sigma) \geq 1 + m_j \geq c_d \cdot m_j + m_j = PC_{t\text{-}LB}^j(\sigma)$.
2. $\underline{j \in X, \text{ and } j \neq j_0}$. Then we know that $c_d \cdot m_{j_i} \leq c_d + 1$. $PC_A^j(\sigma) \geq 1 + c_d + m_j \geq c_d \cdot m_j + m_j = PC_{t\text{-}LB}^j(\sigma)$.
3. $\underline{A \text{ turns the } n^{th} \text{ read of } j \text{ into a saving-read}}$. No matter $j = j_0$ or not, $1 + c_d \geq c_d \cdot m_j$, and $PC_A^j(\sigma) \geq (1 + c_d) \cdot n + 1 + m_j - n \geq 1 + c_d + m_j \geq PC_{t\text{-}LB}^j(\sigma)$.
4. $\underline{A \text{ does not put a copy at } j \text{ at all}}$. $PC_A^j(\sigma) \geq (1 + c_d) \cdot m_j = PC_{t\text{-}LB}^j(\sigma)$.

c: $j \in R \cup K$. For the $2 \cdot r$ different processors in $R \cup K$, we compare their p-cost with pairs of processors $\{i_j, k_j\}$ for $1 \leq j \leq r$. By the definition of t-LB, we know at the write that $\delta(k_j) \leq 0$ since $k_j \notin T$ and $y < t$. Also we see from the definition of t-LB that we can arrange the pairs so that $\delta(i_j) \geq \delta(k_j)$ for all $j = 1, \ldots, r$. Let $PP_j = PC^{i_j}_{t\text{-}LB}(\sigma) + PC^{k_j}_{t\text{-}LB}(\sigma)$ and $PP'_j = PC^{i_j}_{A}(\sigma) + PC^{k_j}_{A}(\sigma)$. We compare PP_j and PP'_j in the following cases.

1. $i_j = j_0$, and A does not put a replica at i_j's local database at all. By $\delta(i_j) \geq \delta(k_j)$, we obtain $c_d \cdot m_{i_j} - 1 \geq c_d \cdot (m_{k_j} - 1) - 1$, hence $m_{i_j} \geq m_{k_j} - 1$. $PP_j = 1 + m_{i_j} + (1 + c_d) \cdot m_{k_j}$; $PP'_j = (1 + c) \cdot m_{i_j} + 1 + c + m_{k_j} \geq m_{i_j} + c_d \cdot (m_{k_j} - 1) + 1 + c_d + m_{k_j} = PP_j$.
2. $i_j = j_0$, and A turns the n^{th} read of i_j into a saving-read, where $(1 \leq n \leq m_{i_j})$. From $\delta(k_j) \leq 0$ we obtain $1 + c_d \geq c_d \cdot m_{k_j}$. $PP_j = 1 + m_{i_j} + (1 + c_d) \cdot m_{k_j}$; $PP'_j = (1 + c_d) \cdot n + 1 + m_{i_j} - n + 1 + c_d + m_{k_j} = 1 + m_{i_j} + c_d \cdot n + 1 + c_d + m_{k_j} \geq 1 + m_{i_j} + c_d \cdot m_{k_j} + m_{k_j} = PP_j$.
3. $k_j = j_0$, and A does not put a replica at i_j's local database at all. By $\delta(i_j) \geq \delta(k_j)$, we obtain $c_d \cdot m_{i_j} - c_d - 1 \geq c_d \cdot m_{k_j} - 1$, hence $m_{i_j} \geq m_{k_j} + 1$. $PP_j = 1 + c_d + m_{i_j} + (1 + c_d) \cdot m_{k_j}$; $PP'_j = (1 + c_d) \cdot m_{i_j} + 1 + m_{k_j} \geq m_{i_j} + c_d \cdot (m_{k_j} + 1) + 1 + m_{k_j} = PP_j$.
4. $k_j = j_0$, and A turns the n^{th} read of i_j into a saving-read, where $(1 \leq n \leq m_{i_j})$. From $\delta(k_j) \leq 0$ we obtain $1 \geq c_d \cdot m_{k_j}$. $PP_j = 1 + c_d + m_{i_j} + (1 + c_d) \cdot m_{k_j}$; $PP'_j = (1 + c_d) \cdot n + 1 + m_{i_j} - n + 1 + m_{k_j} = 1 + c_d \cdot n + m_{i_j} + 1 + m_{k_j} \geq 1 + c_d + m_{i_j} + c_d \cdot m_{k_j} + m_{k_j} = PP_j$.
5. $i_j \neq j_0, k_j \neq j_0$, and A does not put a replica at i_j's local database at all. From $\delta(i_j) \geq \delta(k_j)$, we obtain $c_d \cdot m_{i_j} - c_d - 1 \geq c_d \cdot m_{k_j} - c_d - 1$, hence $m_{i_j} \geq m_{k_j}$. $PP_j = 1 + c_d + m_{i_j} + (1 + c_d) \cdot m_{k_j}$. $PP'_j = (1 + c_d) \cdot m_{i_j} + 1 + c_d + m_{k_j} = 1 + c_d + m_{i_j} + c_d \cdot m_{i_j} + m_{k_j} \geq PP_j$.
6. $i_j \neq j_0, k_j \neq j_0$, and A turns the n^{th} read of i_j into a saving-read, where $(1 \leq n \leq m_{i_j})$. From $\delta(k_j) \leq 0$ we obtain $1 + c_d \geq c_d \cdot m_{k_j}$. $PP_j = 1 + c_d + m_{i_j} + (1 + c_d) \cdot m_{k_j}$; $PP'_j = (1 + c_d) \cdot n + 1 + m_{i_j} - n + 1 + c_d + m_{k_j} = 1 + c_d \cdot n + m_{i_j} + 1 + c_d + m_{k_j} \geq PP_j$.

Thus, in this case we also obtain $\sum_{j=1}^{N} PC^{j}_{t\text{-}LB}(\sigma) \leq \sum_{j=1}^{N} PC^{j}_{A}(\sigma)$.

Summarizing the above two cases, we showed that

$$ST_{t\text{-}LB}(I, \sigma) = \sum_{j=1}^{N} PC^{j}_{t\text{-}LB}(\sigma) \leq \sum_{j=1}^{N} PC^{j}_{A}(\sigma) = ST_A(I, \sigma) \qquad \square$$

A.3.5 For Arbitrary Schedules

Lemma 3.3: Suppose that I is the initial allocation scheme. Suppose that $\psi = \psi_0 \psi_1 \ldots \psi_k$, where ψ_0 is the

first type schedule and ψ_i's are the second type schedules for $i = 1, \ldots, k$. Suppose that A is an arbitrary t-available constrained DOM algorithm. Suppose that for $i = 1, \ldots, k$, the allocation scheme at the i^{th} write of ψ using the algorithm A is I_i, and suppose that the A-allocated schedule is $las_A(\psi) = \xi_0 \ldots \xi_k$, where ξ_i corresponds to ψ_i. Then

$$ST_A(I, \psi) = ST(I, \xi_0) + \sum_{i=1}^{k} ST(I_i, \xi_i) \qquad (3.5.1.1)$$

$$CO_A(I, \psi) = CO(I, \xi_0) + \sum_{i=1}^{k} CO(I_i, \xi_i) \qquad (3.5.1.2)$$

Proof: It follows trivially from the definition of COST in section 2.2. □

Lemma 3.4: Suppose that I is the initial allocation scheme. Suppose that $\psi = \psi_0\psi_1 \ldots \psi_k$, where ψ_0 is the first type schedule and ψ_i's are the second type schedules for $i = 1, \ldots, k$. Suppose J_i is the allocation scheme at the i^{th} write of ψ using the t-LB algorithm. Then

$$ST_{t\text{-}LB}(I, \psi) = ST_{t\text{-}LB}(I, \psi_0) + \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i) \qquad (3.5.2)$$

Proof: Suppose that the t-LB-allocated schedule of ψ is $\xi_0 \ldots \xi_k$, where ξ_i corresponds to ψ_i. ¿From the definition of t-LB we see that the execution sets of the requests in ξ_i depend on the subschedule ψ_i only and are independent of the rest of the schedule. Therefore $ST_{t\text{-}LB}(I, \psi_0) = ST(\xi_0)$ and $ST_{t\text{-}LB}(J_i, \psi_i) = ST(\xi_i)$ for $i = 1, \ldots, k$. Using (3.5.1) we can easily derive (3.5.2). □

A.3.6 The proof of theorem 5

Proof: Suppose that I is the initial allocation scheme. Suppose that ψ is an arbitrary given schedule with k writes, $(k \geq 0)$. Then we can rewrite the schedule ψ as $\psi = \psi_0\psi_1 \ldots \psi_k$, where ψ_0 is the first type schedule and ψ_i's are the second type schedules for $i = 1, \ldots, k$. Suppose that A is an arbitrary t-available constrained DOM algorithm. Suppose that the A-allocated schedule is $las_A(\psi) = \xi_0 \ldots \xi_k$, where ξ_i corresponds to ψ_i. Denote the allocation scheme at the i^{th} write using the t-LB algorithm by J_i. Denote the allocation scheme at the i^{th} write using the algorithm A by I_i. Let $J_0 = I$.

Suppose that for $i = 0, 1, \ldots, k$, $\xi_i = las_{A_i}(\psi_i)$, where A_i is a DOM algorithm which makes the A_i-allocated schedule of ψ_i to be ξ_i for the given initial allocation scheme J_i. ¿From lemma 3.1 we derive $ST_{t\text{-}LB}(I, \psi_0) \leq ST_{A_0}(I, \psi_0) = ST(I, \xi_0)$. Observe that for the subschedules ψ_i $(1 \leq i \leq k)$, the ST-cost $ST(I_i, \xi_i)$ does not depend on the allocation scheme I_i, and $ST(I_i, \xi_i) = ST(J_i, \xi_i) = ST_{A_i}(J_i, \psi_i)$. From lemma 3.2 we obtain $ST_{t\text{-}LB}(J_i, \psi_i) \leq ST_{A_i}(J_i, \psi_i)$ for $i = 1, \ldots, k$. Thus, we have

$$ST_{t\text{-}LB}(I, \psi_0) \leq ST(I, \xi_0) \quad \textit{and} \quad ST_{t\text{-}LB}(J_i, \psi_i) \leq ST(I_i, \xi_i)$$

The theorem follows immediately from the formula (3.5.1.1), (3.5.2) and the inequalities above. □

A.4 The Proof of Theorem 2 and Theorem 3

Following the proof outlined in A.1, we will show in this section that for any input schedule ψ, $ST_{DA}(I,\psi) \leq 2 \cdot ST_{t\text{-}LB}(I,\psi)$, and $CO_{DA}(I,\psi) \leq 2 \cdot c_c \cdot ST_{t\text{-}LB}(I,\psi)$. Additionally, we show that if $c_d > 1$ then $CO_{DA}(I,\psi) \leq c_c \cdot ST_{t\text{-}LB}(I,\psi)$.

In formula (3.5.2), we found a way to split the ST-cost $ST_{t\text{-}LB}(I,\psi)$ into a sum of several ST-costs of special types of subschedules. In this section we will find the lower bounds of each item on the right hand side of (3.5.2). Meanwhile, we will derive similar formulae (similar to (3.5.2)) for the ST-cost $ST_{DA}(I,\psi)$ and CO-cost $CO_{DA}(I,\psi)$ in lemma 4.1 below. And then we find the upper bounds for every item on the right hand side of formulae (4.2.1.1) and (4.2.1.2). At last, we compare the sum of the lower bounds and the sum of the upper bounds to find the competitiveness factor of DA.

Before we proceed, in the next subsection A.4.1 we give some notations which will be used through A.4. Then in section A.4.2, we give some auxiliary lemmas to estimate the bounds of each piece on the right hand side of (3.5.2), (4.2.1.1) and (4.2.1.2). Lastly in section A.4.3, we compare the sum of the bounds and show the theorems 2 and 3.

A.4.1 Notations

We assume that the initial allocation scheme $I = F \cup \{p\}$ is of size t. For an arbitrary given schedule with k writes, $k \geq 0$, we re-write the schedule as $\psi = \psi_0\psi_1 \ldots \psi_k$, where ψ_0 is zero or more read requests which come before the first write, and ψ_i is the i^{th} write w^{p_i} followed by zero or more reads which come before the next write for $1 \leq i \leq k$. Let $p_0 = p$. For $0 \leq i \leq k$, we denote by

a_i : the number of different reading processors of ψ_i which are not in $F \cup \{p_i\}$;

b_i : the number of different reading processors of ψ_i;

s_i : the number of read requests in ψ_i.

Then, obviously we have $0 \leq a_i \leq b_i \leq s_i$ for $0 \leq i \leq k$.

Denote by I_i the allocation scheme at w^{p_i} using the DA algorithm, and denote by J_i the allocation scheme at w^{p_i} using the t-LB algorithm.

A.4.2 Bound Estimation

Lemma 4.1:

$$ST_{DA}(I,\psi) = ST_{DA}(I,\psi_0) + \sum_{i=1}^{k} ST_{DA}(I_i,\psi_i) \qquad (4.2.1.1)$$

$$CO_{DA}(I,\psi) = CO_{DA}(I,\psi_0) + \sum_{i=1}^{k} CO_{DA}(I_i,\psi_i) \qquad (4.2.1.2)$$

Proof: Suppose that the DA-allocated schedule of ψ is $\xi_0\xi_1\ldots\xi_k$, where ξ_i corresponds to ψ_i.

Observe that in the COST definition of a request and the definition of DA, the execution set of a read request in the DA-allocated schedule and its COST depend only on its current status (1 for data processor, and $2+c_c+c_d$ for non data processor); And the execution set of a write and its COST depend only on the writing processor itself and the allocation scheme at the write. Therefore, $las_{DA}(\psi_0) = \xi_0$, $COST(I,\xi_0) = COST_{DA}(I,\psi_0)$. And for the schedule ψ_i with the given initial allocation scheme I_i, the DA-allocated schedule $las_{DA}(\psi_i) = \xi_i$ for $i = 1,\ldots,k$, and $COST(I_i,\xi_i) = COST_{DA}(I_i,\psi_i)$. Thus, the formulae (4.2.1.1) and (4.2.1.2) follows from (3.5.1.1), (3.5.1.2) and (1.1). □

Next, in terms of the notations given in A.4.1, we estimate $ST_{DA}(I,\psi_0)$, $CO_{DA}(I,\psi_0)$ in lemma 4.2; find the lower bound of $ST_{t\text{-}LB}(I,\psi_0)$ in lemma 4.3. Then we give a lower bound of $ST_{t\text{-}LB}(J_i,\psi_i)$ in lemma 4.4, and an upper bound of $CO_{DA}(I_i,\psi_i)$ in lemma 4.5. In lemma 4.6 we show that $ST_{DA}(I_i,\psi_i)$ is at most twice as much as $ST_{t\text{-}LB}(J_i,\psi_i)$.

Lemma 4.2:

$$ST_{DA}(I,\psi_0) = (1+c_d)\cdot a_0 + s_0 \qquad (4.2.2.1)$$

$$CO_{DA}(I,\psi_0) = c_c \cdot a_0 \qquad (4.2.2.2)$$

Proof: There are a_0 different reading processors which are not in the initial allocation scheme I, and the first read from each of these processors in ψ_0 will cost $(c_c + c_d + 1)$ more than the other reads. c_c counts for the cost of submitting the read request to a data processor; c_d counts for the cost of transmitting the object; and 1 counts for the cost of the saving I/O operation, namely, outputting the object to the local database. Each read request costs 1 for inputting the object from a local database. Therefore, the total the total ST-cost is $(1+c_d)\cdot a_0 + s_0$ and the CO-cost is $c_c \cdot a_0$. □

Lemma 4.3:

$$ST_{t\text{-}LB}(I,\psi_0) \geq a_0 \cdot c_d + s_0 \qquad (4.2.3)$$

Proof: The object has to be transmitted to the a_0 non data processors for the read requests, hence the object transmitting cost is at least $a_0 \cdot c_d$. Each read request has to be satisfied by inputting the object from

a local database, hence the I/O cost is at least s_0. The lemma follows. □

Lemma 4.4: For $i = 1, \ldots, k$;

$$ST_{t\text{-}LB}(J_i, \psi_i) \geq t + a_i \cdot c_d + s_i \qquad (4.2.4)$$

Proof: The I/O cost of the i^{th} write is t, and the I/O cost of the s_i reads is at least s_i. Since there are at least a_i different reading processors which are not the writing processor p_i, the object transmitting cost is at least $a_i \cdot c_d$. The lemma follows. □

Lemma 4.5: For $i = 1, \ldots, k$;

$$CO_{DA}(I_i, \psi_i) \leq (1 + a_{i-1} + b_i) \cdot c_c \qquad (4.2.5)$$

Proof: At the write w^{p_i}, there are at most $(1+a_{i-1})$ data processors which are not in the execution set, 1 counts for p_{i-1} and a_{i-1} counts for the processors which belong to I_i due to a read. Thus, the CO-cost of the 'invalidating' message is at most $(1 + a_{i-1}) \cdot c_c$.

Besides, there are b_i different reading processors in ψ_i, these processors either have a replica after the write (hence involve no control message cost), or keep a replica after their first reads (with a single control message cost c_c). Therefore, the total control message cost of the reads are at most $b_i \cdot c_c$.

Thus, the total control message cost of ψ_i with the initial allocation scheme I_i using the DA algorithm is at most $(1 + a_{i-1} + b_i) \cdot c_c$. □

Lemma 4.6: For $i = 1, \ldots, k$;

$$ST_{DA}(I_i, \psi_i) \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i) \qquad (4.2.6)$$

Proof : Denote the set of reading processors of ψ_i by B. Then $|B| = b_i$. Suppose that T is the execution set of w^{p_i} using the t-LB algorithm, and Y is the set of processors j which satisfy $\delta(j) > 0$ at w^{p_i}. Let $y = |Y|$. Denote by m_j the number of reads issued from processor j in ψ_i. Then, $\sum_{j \in B} m_j = s_i$.

To derive the formula (4.2.6), we analyze the $ST_{DA}(I_i, \psi_i)$ and $ST_{t\text{-}LB}(J_i, \psi_i)$ in the following cases, namely, I) $y \geq t$ and $p_i \in Y$; II) $y \geq t$ and $p_i \notin Y$; III) $y < t$ and $p_i \in Y$; and IV) $y < t$ and $p_i \notin Y$.

Case I : $\underline{y \geq t \text{ and } p_i \in Y}$. Then $p_i \in T \subset Y$, $t \leq y \leq b_i$.

Let's consider the lower bound of $ST_{t\text{-}LB}(J_i, \psi_i)$ first. The ST-cost of the write is $t + (t-1) \cdot c_d$; For the processors $j \in T$, the total ST-cost of the reads from j is m_j for the local I/O; For the processors $j \in Y/T$, the total ST-cost of the reads from j is $1 + c_d + m_j$, where c_d is the cost of transmitting the

object form a processor in F to j for the first read of j, 1 is the cost of outputting the object to the local database of j for the saving-read (the first read of j will be turned into a saving-read), m_j is the inputting I/O cost for retrieving the object; For the processors $j \in B/T$, j will not have a replica of the object at all, the total ST-cost of the reads form j is $(1 + c_d) \cdot m_j$ since every read will be a remote read. Therefore,

$$ST_{t\text{-}LB}(J_i, \psi_i) = t + (t-1) \cdot c_d + \sum_{j \in T} m_j + \sum_{j \in Y/T} (1 + c_d + m_j) + \sum_{j \in B/Y} (1 + c_d) \cdot m_j$$

$$= t + (t-1) \cdot c_d + (1 + c_d) \cdot (y - t) + \sum_{j \in B} m_j + \sum_{j \in B/Y} c_d \cdot m_j$$

$$\geq_{(m_j \geq 1)} y + (y-1) \cdot c_d + s_i + c_d \cdot (b_i - y) = y + s_i + c_d \cdot (b_i - 1)$$

Now consider $ST_{DA}(I_i, \psi_i)$. The write costs $t + (t-1) \cdot c_d$; The reads from p_i are all local reads, hence costs m_{p_i}; The reads from other processors $j \in B/\{p_i\}$ costs at most $1 + c_d + m_j$. Therefore,

$$ST_{DA}(I_i, \psi_i) \leq t + (t-1) \cdot c_d + m_{p_i} + \sum_{j \in B/\{p_i\}} (1 + c_d + m_j) =$$

$$t + (t-1) \cdot c_d + \sum_{j \in B} m_j + (b_i - 1) \cdot (1 + c_d) = t + s_i + b_i - 1 + (t - 1 + b_i - 1) \cdot c_d$$

Let $A_1 = y + s_i$, $A_2 = b_i - 1$, $B_1 = t + s_i + b_i - 1$, and $B_2 = t - 1 + b_i - 1$. Since $t \leq y \leq b_i \leq s_i$, we derive $B_1 \leq 2 \cdot A_1$ and $B_2 \leq 2 \cdot A_2$. Thus,

$$ST_{DA}(I_i, \psi_i) \leq B_1 + B_2 \cdot c_d \leq 2 \cdot (A_1 + A_2 \cdot c_d) \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i)$$

Case II : $\underline{y \geq t \text{ and } p_i \notin Y}$. Then $p_i \notin T$, $T \subset Y$, $t \leq y \leq b_i \leq s_i$, and similarily (to case I) we get

$$ST_{t\text{-}LB}(J_i, \psi_i) = t + t \cdot c_d + \sum_{j \in T} m_j + \sum_{j \in Y/T} (1 + c_d + m_j) + \sum_{j \in B/Y} (1 + c_d) \cdot m_j =$$

$$y + y \cdot c_d + \sum_{j \in B} m_j + \sum_{j \in B/Y} c_d \cdot m_j \geq y + y \cdot c_d + s_i + c_d \cdot (b_i - y) = y + s_i + c_d \cdot b_i$$

$$ST_{DA}(I_i, \psi_i) \leq t + (t-1) \cdot c_d + \sum_{j \in B} (1 + c_d + m_j) =$$

$$t + (t-1) \cdot c_d + s_i + (1 + c_d) \cdot b_i = t + s_i + b_i + c_d \cdot (t - 1 + b_i)$$

We can easily derive $t + s_i + b_i \leq 2 \cdot (y + s_i)$, and $t - 1 + b_i \leq 2 \cdot b_i$ in this case. Thus, we derive the formula (4.2.6) from the two bounds obtained.

Case III : $\underline{y < t \text{ and } p_i \in Y}$. Then $Y \subset T$. We analyze two subcases 1) $b_i \geq t$; and 2) $b_i < t$. In each case we follow the same analysis as we did in the previous cases to find the lower bound of $ST_{t\text{-}LB}(J_i, \psi_i)$ and the upper bound of $ST_{DA}(I_i, \psi_i)$, and then we compare the bounds.

1. $\underline{b_i \geq t}$. Then $T \subset B$ and

$$ST_{t\text{-}LB}(J_i, \psi_i) = t + (t-1) \cdot c_d + \sum_{j \in T} m_j + \sum_{j \in B/T} (1 + c_d) \cdot m_j$$

$$\geq t + (t-1) \cdot c_d + s_i + c_d \cdot (b_i - t) = t + s_i + (b_i - 1) \cdot c_d$$

$$ST_{DA}(I_i, \psi_i) \leq t + (t-1) \cdot c_d + m_{p_i} + \sum_{j \in B/\{p_i\}} (1 + c_d + m_j)$$

$$= (1 + a_{i-1}) \cdot c_c + t + (t-1) \cdot c_d + s_i + (b_i - 1) \cdot (c_c + 1 + c_d) =$$

$$t + s_i + b_i - 1 + (t - 1 + b_i - 1) \cdot c_d \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i)$$

2. $\underline{b_i < t}$. Then $B \subset T$, and

$$ST_{t\text{-}LB}(J_i, \psi_i) = t + (t-1) \cdot c_d + \sum_{j \in B} m_j = t + s_i + (t-1) \cdot c_d$$

$$ST_{DA}(I_i, \psi_i) \leq t + (t-1) \cdot c_d + m_{p_i} + \sum_{j \in B/\{p_i\}} (1 + c_d + m_j)$$

$$= t + s_i + b_i - 1 + (t - 1 + b_i - 1) \cdot c_d \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i)$$

Thus, (4.2.6) is true in this case.

Case IV : $\underline{y < t \text{ and } p_i \notin Y}$. Then $Y \subset T$. We analyze three subcases, namely, 1) $b_i \geq t$; 2) $0 < b_i < t$; and 3) $b_i = 0$. In each of these cases, we find the lower bound of $ST_{t\text{-}LB}(J_i, \psi_i)$ and upper bound of $ST_{DA}(I_i, \psi_i)$, and then compare the bounds.

1. $\underline{b_i \geq t}$. Then $Y \subset T \subset B$, and

$$ST_{t\text{-}LB}(J_i, \psi_i) = t + t \cdot c_d + \sum_{j \in T} m_j + \sum_{j \in B/T} (1 + c_d) \cdot m_j = t + t \cdot c_d + s_i + c_d \cdot \sum_{j \in B/T} m_j$$

$$\geq t + s_i + t \cdot c_d + c_d \cdot (b_i - t) = t + s_i + c_d \cdot b_i$$

$$ST_{DA}(I_i, \psi_i) \leq t + (t-1) \cdot c_d + \sum_{j \in B} (1 + c_d + m_j) =$$

$$t + s_i + b_i + (t - 1 + b_i) \cdot c_d \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i)$$

2. $\underline{0 < b_i < t}$. Then $B \subset T$, and

$$ST_{t\text{-}LB}(J_i, \psi_i) = t + (t-1) \cdot c_d + \sum_{j \in B} m_j = t + s_i + (t-1) \cdot c_d$$

$$ST_{DA}(I_i, \psi_i) \leq t + (t-1) \cdot c_d + \sum_{j \in B} (1 + c_d + m_j) =$$

$$t + s_i + b_i + (t - 1 + b_i) \cdot c_d \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i)$$

3. $\underline{b_i = 0}$. Then $B = \emptyset$, and

$$ST_{t\text{-}LB}(J_i, \psi_i) = ST_{DA}(I_i, \psi_i) = t + (t-1) \cdot c_d$$

Obviously, (4.2.6) follows in this case. □

A.4.3 Bounds Comparison

In this section, we show firstly that with the given initial allocation scheme I the ST-cost of DA-allocated schedule of an arbitrary schedule ψ is at most twice as much as the optimum ST-cost (i.e., the ST-cost of the corresponding t-LB-allocated schedule). Secondly, we show that the CO-cost of DA-allocated schedule of an arbitrary schedule is at most $2 \cdot c_c$ (or c_c if $c_d > 1$) times as much as the optimum ST-cost. Then we use the arguments in A.1 to prove the theorems.

Lemma 4.7:

$$ST_{DA}(I, \psi) \leq 2 \cdot ST_{t\text{-}LB}(I, \psi) \tag{4.3.1}$$

Proof: From (4.2.2.1) we see that $ST_{DA}(I, \psi_0) = a_0 + c_d \cdot a_0 + s_0$. Since $a_0 \leq s_0$, from (4.2.3) we can easily conclude $ST_{DA}(I, \psi_0) \leq 2 \cdot ST_{t\text{-}LB}(I, \psi_0)$. ¿From this inequality and (4.2.6) we see that

$$ST_{DA}(I, \psi_0) + \sum_{i=1}^{k} ST_{DA}(J_i, \psi_i) \leq 2 \cdot ST_{t\text{-}LB}(I, \psi_0) + 2 \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

From (4.2.1.1), (3.5.2) and the inequality just obtained, we derive (4.3.1). □

Proof of Theorem 2: Since $t > 1$ and $a_i \leq b_i \leq s_i$, we derive $1 + a_i + b_i \leq 2 \cdot (1 + s_i) \leq 2 \cdot (t + a_i \cdot c_d + s_i) \leq_{(4.2.4)}$ $2 \cdot ST_{t\text{-}LB}(J_i, \psi_i)$. Therefore, for $i = 1, \ldots, k$,

$$1 + a_i + b_i \leq 2 \cdot ST_{t\text{-}LB}(J_i, \psi_i) \tag{4.3.2}$$

By summing the inequality (4.2.5) up from $i = 1$ to $i = k$, we obtain

$$\sum_{i=1}^{k} CO_{DA}(I_i, \psi_i) \leq c_c \cdot \sum_{i=1}^{k} (1 + a_{i-1} + b_i) \leq c_c \cdot a_0 + c_c \cdot \sum_{i=1}^{k} (1 + a_i + b_i)$$

Thus, from (4.3.2) we obtain

$$\sum_{i=1}^{k} CO_{DA}(I_i, \psi_i) \leq c_c \cdot a_0 + 2 \cdot c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

By adding (4.2.2.2) to the inequality above, we get

$$CO_{DA}(I, \psi_0) + \sum_{i=1}^{k} CO_{DA}(I_i, \psi_i) \leq 2 \cdot c_c \cdot a_0 + 2 \cdot c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

$$\leq_{(4.2.3)} 2 \cdot c_c \cdot ST_{t\text{-}LB}(I, \psi_0) + 2 \cdot c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

Thus, by (4.2.1.2) and (3.5.2) we conclude

$$CO_{DA}(I, \psi) \leq 2 \cdot c_c \cdot ST_{t\text{-}LB}(I, \psi) \tag{4.3.3}$$

By adding (4.3.1) to this inequality, from (1.1) we derive

$$COST_{DA}(I, \psi) \leq (2 + 2 \cdot c_c) \cdot ST_{t\text{-}LB}(I, \psi) \leq_{(1.2)} (2 + 2 \cdot c_c) \cdot COST_{OPT}(I, \psi) \qquad \square$$

Proof of Theorem 3: Since $t > 1$, $c_d > 1$ and $s_i \geq b_i$, from (4.2.4) we derive that for $i = 1, \ldots, k$,

$$ST_{t\text{-}LB}(J_i, \psi_i) \geq 1 + a_i + b_i \tag{4.3.4}$$

$$2 \cdot a_0 \leq a_0 \cdot c_d + s_0 \tag{4.3.5}$$

Now we sum the inequality (4.2.5) up from $i = 1$ to $i = k$. Then we obtain

$$\sum_{i=1}^{k} CO_{DA}(I_i, \psi_i) \leq c_c \cdot \sum_{i=1}^{k} (1 + a_{i-1} + b_i) \leq c_c \cdot a_0 + c_c \cdot \sum_{i=1}^{k} (1 + a_i + b_i)$$

Thus from (4.3.4) we obtain

$$\sum_{i=1}^{k} CO_{DA}(I_i, \psi_i) \leq c_c \cdot a_0 + c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

By adding (4.2.2.2) to the inequality above, we get

$$CO_{DA}(I, \psi_0) + \sum_{i=1}^{k} CO_{DA}(I_i, \psi_i) \leq 2 \cdot c_c \cdot a_0 + c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

$$\leq_{(4.3.5)} c_c \cdot (a_0 \cdot c_d + s_0) + c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

$$\leq_{(4.2.3)} c_c \cdot ST_{t\text{-}LB}(I, \psi_0) + c_c \cdot \sum_{i=1}^{k} ST_{t\text{-}LB}(J_i, \psi_i)$$

¿From (4.2.1.2) and (3.5.2) and the inequality just obtained, we conclude

$$CO_{DA}(I,\psi) \leq c_c \cdot ST_{t\text{-}LB}(I,\psi) \qquad (4.3.6)$$

By adding (4.3.1) to (4.3.6), from (1.1) we derive

$$COST_{DA}(I,\psi) \leq (2+c_c) \cdot ST_{t\text{-}LB}(I,\psi) \leq_{(1.2)} (2+c_c) \cdot COST_{OPT}(I,\psi) \quad \square$$