# Implementing On-line Techniques to Allocate File Resources in Large Distributed Systems*

Elena Pagani, Gian Paolo Rossi
Dip. di Scienze dell'Informazione - Università degli Studi di Milano
e-mail: {pagae,rossi}@dsi.unimi.it

## Abstract

*The allocation of shared resources in a distributed system is a key aspect to achieve both low latency in accessing the resource and low bandwidth consumption. When the set of users accessing a resource dynamically changes, the allocation policy should adapt the resource placement over time. In the literature, on-line algorithms have been proposed that dynamically relocate a file in the centre of gravity of the set of users that are more frequently accessing it. However, so far those algorithms have not been implemented. They must be adapted to work in real systems, and their interactions must be investigated with the existing network protocols and applications.*

*In this work we study the behaviours of some of those algorithms in real environments. To this purpose, we devised an implementation scheme, and we measured the algorithms performance in real settings.*

## 1. Introduction

The growing diffusion of the Internet and of network-based applications is favouring the deployment of large information systems. Those applications are characterized by a large amount of users, either sparsely or densely located in the network, and whose membership dynamically changes. To achieve both high efficiency in the use of the network resources (i.e., the bandwidth), and low latency in accessing the information under these conditions, it is necessary to place the files containing the information *close* to the users that more frequently access them. Unfortunately, the set of users could be not precisely characterized, or it may change over time. This problem will be emphasized by the emerging QoS applications, that have peculiar requirements in terms of bandwidth availability and low data transmission latency.

*This work was supported by the MURST under Contract no.9809321920, "Transport of multicast packets with QoS guarantees".

In the field of *on-line computation* some algorithms have been proposed, that *dynamically* relocate the files. They try to characterize the current *access locality*, that is the set of processors that at a given time are more frequently accessing the file, and they move the file to the centre of gravity of that locality. Those algorithms aim at finding a *trade-off* between the access costs paid by a user to access the file, and the management costs paid by the system to relocate that file. Both costs are defined in terms of communication costs. The algorithms seem promising to reduce the amount of exchanged messages, and in some cases they effectively approximate the optimal allocation strategy.

So far, the on-line algorithms have been studied mostly under a theoretical point of view. Their performance has been evaluated in the worst case, using the *competitive analysis*. In this work we study the behaviours of some of those algorithms in a real environment, where additional practical considerations and constraints have to be addressed. To this purpose, we devised an implementation scheme, and we measured the algorithm performance in realistic conditions, by means of simulation techniques. Our experimental results allow to highlight pros and cons of the different approaches for file allocation, and to characterize both under what conditions and to what extent the use of a dynamic allocation algorithm is advantageous.

The paper is organized as follows: in section 2, we introduce the network model and the definition of the considered problems. In section 3, we describe the studied algorithms and we discuss the issues related with their implementation in real networks. In section 4, we describe the implementation scheme adopted for the considered algorithms. In section 5, we report the experimental results obtained by measuring the performance of the implemented algorithms in real settings.

## 2. Network model

A network $\mathcal{N}$ can be modeled as an undirected weighted graph $G = \langle V, E \rangle$, where the processors are represented by the vertices, while the edges represent the links connecting the nodes. The weight of an edge represents the cost to transmit a data packet over the corresponding link. The weight function $w : E \to \mathbb{R}$ may account for the link length, or the link bandwidth or other quality-of-link metrics. We indicate with $w(p, q)$ the weight of a path connecting two processors $p$ and $q$.

Without loss of generality, we consider the problem of allocating a unique file; different files are allocated independently. We denote by $\mathcal{R}$, $\mathcal{R} \subseteq V$, the set of processors owning a file copy. $\mathcal{R}$ can dynamically change over time.

We denote with $\sigma$ the sequence of requests to access the file. Let $\mathcal{G}$, $\mathcal{G} \subseteq V$, be the group of processors accessing the file. We say that $\sigma(i) = p$, with $p \in \mathcal{G}$, if the $i$-th request has been issued by processor $p$. For our purposes, it is not relevant the order in which two concurrently generated requests appear in $\sigma$.

Let $f_c$ be the cost function; $f_c$ may refer either to the cost to access the file for a user, or to the *management* cost, that is, the cost to modify the allocation scheme. Both costs are evaluated in terms of exchanged messages.

The access cost is usually measured by differentiating between *read* and *write* operations. In a *uniform* network, that is, a network where $\forall p, q \in V \; w(p, q) = 1$, we define $f_c$ for the access costs as follows:

- we assume that a read request only requires one packet to be sent by the requesting processor. As the consequence: $f_c(read) = 1$;
- a write request updates the file content. To maintain the consistency among the file replicas, the write operation must be performed on all the replicas. As the consequence: $f_c(write) = |\mathcal{R}|$.

Let us notice that $f_c(write) = f_c(read)$ when $|\mathcal{R}| = 1$. Hence, when only one replica is maintained in the system, we do not distinguish between the two operations, but rather we refer to a generic operation *op* of accessing the file, whose cost is $f_c(op) = 1$.

In non-uniform networks, we denote with $d(p, q)$ the weight of the best path (according to the considered quality-of-path metrics) connecting $p$ to $q$; that is, $d(p, q) = \min_{\forall paths\ p \to q}\{w(p, q)\}$. Then, the $f_c$ definition becomes:

- $f_c(read) = \min_{\forall r \in \mathcal{R}} d(p, r)$, where $p$ is the processor generating the read request. This is the cost paid by $p$ to access the nearest file replica;
- $f_c(write) = \min_{\forall r \in \mathcal{R}} d(p, r) + \sum_{\forall r' \in \mathcal{R}, r' \neq r} d(r, r')$.

Also in this case, $f_c(read) = f_c(write)$ when $|\mathcal{R}| = 1$, and $f_c(op) = d(p, r)$ with $r$ the file holder. In both uniform and non-uniform networks, a processor in $\mathcal{R}$ can access its own file replica at no cost, because no messages are generated. $f_c(\sigma)$ is the sum of all the access costs paid to satisfy $\sigma$.

Clearly, the decision of maintaining the replicas on (a subset of) the processors in $\mathcal{G}$ helps to reduce the global cost. Moreover, the processors not interested in accessing the file are not burdened with file maintenance tasks. In this work, we assume that $\mathcal{R} \subseteq \mathcal{G}$.

The operations that may be performed over the file copies are: creation of a new replica ($new$), deletion of an existing replica ($del$), or movement of a replica from a processor $r \in \mathcal{R}$ to a processor $r'$ currently not in $\mathcal{R}$ ($move$). All of those operations dynamically change the $\mathcal{R}$ membership. A $move$ operation updates the $\mathcal{R}$ membership so that $r$ is no longer in $\mathcal{R}$ while $r'$ becomes a member of $\mathcal{R}$. The creation of a replica $r'$ differs from a movement in that it is performed by copying the file from the $r \in \mathcal{R}$ nearest to $r'$. After the creation both $r$ and $r'$ are in $\mathcal{R}$.

Let $F$ be the file size in terms of number of packets needed to transfer the file. We assume that $f_c(del) = 0$, while $f_c(new) = f_c(move) = F$ in uniform networks. In non-uniform networks, $f_c(del) = 0$, while $f_c(new) = f_c(move) = F \times \min_{\forall r \in \mathcal{R}} d(r, r')$, where $r'$ is the processor holding the new replica. These costs represent the management costs.

### 2.1. Problem definition

In the on-line literature a family of problems has been defined, concerning the problem of allocating and maintaining a shared file in distributed systems [5, 6, 3]. Those problems differ in the number of replicas of the shared file maintained and in the operations allowed in accessing a replica. In this work we focus on the two problems of *file migration* ($\mathcal{FM}$) [9, 5] and *file allocation* ($\mathcal{FA}$) [7, 5].

**File migration.** Only one copy of the file exists in the system. This copy may be relocated to minimize the global costs for the processors in $\mathcal{G}$. In this setting no distinction exists between read and write operations.

**File allocation.** More copies of the file can exist in the system; their number must satisfy $1 \leq |\mathcal{R}| \leq |V|$. Read and write operations are distinct. Both $\mathcal{R}$ and $|\mathcal{R}|$ vary depending on the access pattern, that is, the origin and type of the generated access requests, to minimize the global costs. Let us notice that:

- the higher the number of replicas, the lower the mean distance between a processor in $\mathcal{G}$ and a replica in $\mathcal{R}$. Hence, the lower the read cost;
- on the other hand, the higher the number of replicas, the higher the cost to be paid to update them all upon a write request.

## 3. On-line algorithms

The problems defined in the previous section have been studied in the frame of the on-line computation [11, 14]. The on-line algorithms learn the input data *during the run*. For each input, the algorithm must produce an output that only depends on the inputs seen so far. The algorithm performance is evaluated by comparing it with the performance achieved for the same input sequence by an *adversary*.

The *competitive ratio* [8] of an algorithm $\mathcal{A}$ against an *adversary* algorithm $\mathcal{O}$ is defined as: $\sup_\sigma \frac{f_c^{\mathcal{A}}(\sigma+management)}{f_c^{\mathcal{O}}(\sigma+management)}$. An algorithm $\mathcal{A}$ is *c-competitive* if a constant $a$ exists so that, for every input sequence $\sigma$: $f_c^{\mathcal{A}}(\sigma + management) \leq c \cdot f_c^{\mathcal{O}}(\sigma + management) + a$. The lower the competitive ratio is, the better $\mathcal{A}$ approximates the optimal strategy; the adversary's objective is to maximize $c$. A lower bound of 3 has been determined on the competitive ratio for algorithms solving either the $\mathcal{FM}$ or the $\mathcal{FA}$ problem.

Several on-line algorithms have been proposed to solve the $\mathcal{FM}$ and $\mathcal{FA}$ problems [5]. Among them, in this work we study the *Move-to-Min* (MTM) [2], *Move-to-Local-Min* (MTLM) [6] and *Steiner-Based* (SB) [7] algorithms. They adopt the same mechanism to bound the management costs, consisting in reconfiguring the allocation scheme with a rate inversely proportional to $F$. We review them here briefly.

**Move-to-Min algorithm.** This is a deterministic algorithm that solves the $\mathcal{FM}$ problem. The sequence of requests is partitioned into subsequences (*phases*) of length $F$. During a phase the file is not moved. At the end of each phase the file is possibly moved to minimize the access costs for that phase, under the hypothesis that the same request subsequence repeats. Hence, file relocation tries to map the access locality of the previous phase. Let us indicate with $\sigma_i$, $\sigma_i \subset \sigma$, the subsequence of requests issued in the phase $i$; the file is moved to the processor that realizes:

$$\min_{\forall q \in \mathcal{G}} \sum_{p \in \sigma_i} d(p, q).$$

It has been proved [2] that this algorithm has a competitive ratio 7 with arbitrary network topologies.

EXAMPLE: Let us consider the network shown in figure 1($a$), where all edges have weight 1, and let us suppose $F = 11$. Let $\sigma_k = B, C, D, C, C, D, B, E, C, B, G$ be the subsequence of access requests in the $k$-th phase, during which the file is placed on the processor $H$. In figure 1($b$) we report the values of the function to minimize, for all the processors. According to the MTM algorithm, the file owner for the $(k + 1)$-st phase is processor $C$. The cost incurred by the algorithm to move the file is 22. The cost for satisfying $\sigma_k$ has been 23. If the same subsequence repeats in phase $(k + 1)$, the access cost is 10.

**Move-to-Local-Min algorithm.** This algorithm improves the performance of the MTM algorithm. As MTM, it is a deterministic algorithm that solves the $\mathcal{FM}$ problem. MTLM differs from MTM in two respects:

1. the phase length equals $c \cdot F$ for a constant $c > 1$;
2. at the end of a phase the file is migrated onto the processor that realizes

$$\min_{\forall q \in \mathcal{G}} \Big(\sum_{p \in \sigma_i} d(p, q) + \delta \cdot d(r, q) \cdot F\Big)$$

with $r$ the current file owner, and $0 < \delta < 1$.

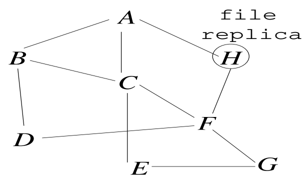
As the consequence, MTLM tries to minimize the global cost of maintaining the file by considering both the *access costs* and the *management costs*. It has been proved [6] that, for $c = 1.841$ and $\delta = 0.648$ the algorithm achieves its best competitive ratio, equal to 4.086.

EXAMPLE: Let us consider the same example seen above. This time the phase length is $c \cdot F \simeq 20$. Let us suppose that the request subsequence is $\sigma_k = B, C, D, C, C, D, B, E, C, B, G, B, C, D, C, C, D, B, E, C$. In figure 1($c$) we report the values of the function to minimize, for all the processors. According to the MTLM algorithm, the file is moved to processor $C$. The cost to satisfy $\sigma_k$ has been 42; if the same subsequence repeats in the phase $(k + 1)$ the cost is 17.

Both MTM and MTLM try to maintain the file in the centre of gravity of the current access locality. But, they decide for the future basing on the past, thus possibly reacting in late.

**Steiner-Based algorithm.** This is a randomized algorithm to solve the $\mathcal{FA}$ problem. It uses as a subroutine an algorithm for the on-line *Steiner Tree* problem [7]. The ST problem consists in incrementally building a minimum spanning tree according to the requests of joining the tree that are issued during the algorithm run. When a join request is generated, the ST algorithm must find the branch to add to the current tree such that the global tree weight is minimized.

The SB algorithm starts with only one copy of the file. When a read request is issued by the processor $p \in \mathcal{G}$, it is satisfied by the nearest processor in $\mathcal{R}$. Then, with probability $1/F$ a join request for $p$ is submitted to the ST subroutine. File replicas are created on the processors belonging to the new tree branch. The tree weight minimization aims at bounding the number of replicas. When a write request is issued by the processor $p$, it is satisfied and then, with probability $1/\alpha F$ a system reconfiguration is started. All the file copies are deleted and the file is only maintained at $p$. Then, a new run of the ST algorithm is started with $p$ as initial configuration.




$(a)$

| | f_MTM |
|---|---|
| A | 16 |
| B | 11 |
| C | 10 |
| D | 16 |
| E | 17 |
| F | 15 |
| G | 22 |
| H | 23 |

$(b)$

| | f_MTLM |
|---|---|
| A | 35.128 |
| B | 33.256 |
| C | 31.256 |
| D | 43.256 |
| E | 52.384 |
| F | 34.128 |
| G | 55.256 |
| H | 42 |

$(c)$

**Figure 1. ($a$) Example network topology. ($b$) Minimizing function for MTM. ($c$) Minimizing function for MTLM.**

It has been proved that, for $\alpha = \sqrt{3}$, the SB algorithm achieves the best performance. If the ST subroutine is $c$-competitive, then the SB algorithm has a competitive ratio $(2+\sqrt{3})c$ against an adaptive on-line adversary. In [12] a greedy algorithm is proposed for the ST problem, which is $(\log n)$-competitive, with $n = |V|$.

EXAMPLE: Let us consider the same example seen above, with all the requests being read requests. For each request, there is a probability $1/11$ that the file allocation scheme is re-configured after that request. Let us notice that the re-configuration decision does *not* depend on the history of access requests, differently from the MTM and MTLM algorithms. For instance, file copies could be created in $F$ and $G$ (issuing the 11-th request), although so far the most part of the accesses have been performed by $C$.

### 3.1. Discussion

The purpose of the on-line algorithms is to characterize policies that achieve good performance in the worst case. To evaluate the algorithms behaviour, they have been analyzed under simplifying assumptions, that are not met by real networks [1]. Indeed, the algorithms proposed in the frame of on-line computation assume that global knowledge about the system state is available. In particular, the complete knowledge of the network topology, the request subsequence and the current replica locations is required.

In real networks, such knowledge can be achieved only by means of the exchange of control messages among the processors, reporting the state of each processor and its knowledge about the system. The cost to diffuse those control messages is not taken into account by the competitive analysis for the theoretical algorithms, whose real performance could then be worse than that estimated by the competitive ratio.

In [7], a possible implementation of the SB algorithm is outlined. This implementation is quite complex under the computational point of view. Moreover, the competitive analysis is used to evaluate the behaviour of the obtained distributed protocol in the worst case, while no evaluation is performed concerning the performance that the protocol could offer in a real system. In the following sections we propose a possible implementation of the considered on-line algorithms. We study their average case behaviours under different network conditions, using simulation techniques.

## 4. Implementation of the on-line algorithms

To implement the on-line algorithms in a real network, they must be integrated with other protocols that provide them with the needed information about the system. In the architecture we implemented, distributed applications exploit the services of the on-line algorithms to access the shared files in the network. Access requests generated by the applications are passed to the on-line algorithm, that takes in charge their forwarding to the nearest replica, and then to all the other replicas in the case of write requests. The on-line algorithm in turn lies on unicast and multicast routing protocols. The unicast routing service supplies the distance to every other processor; it is used to individuate the nearest replica. Each processor in $\mathcal{G}$ sends its own requests to the nearest replica via unicast messages. The multicast routing protocol builds and maintains a *shared tree* infrastructure [10, 4] connecting all the processors in $\mathcal{G}$. Let us point out that the tree characterized by the ST subroutine is a different structure from the multicast routing tree. The former is an abstract structure, only computed to find out on what processors the file must be replicated. The latter is used to route multicast messages among the processors in $\mathcal{G}$. The tree *core* (i.e., the root) node gathers the system state information and forwards it along the tree. Each node sends to the core its unicast routing table either at the system initialization or when the table contents change. This way, the core node learns the whole network topology.

In figure 2($a$) we show the communication pattern for the MTM and MTLM algorithms. The current file holder $r$ receives and processes the requests of accessing the file.

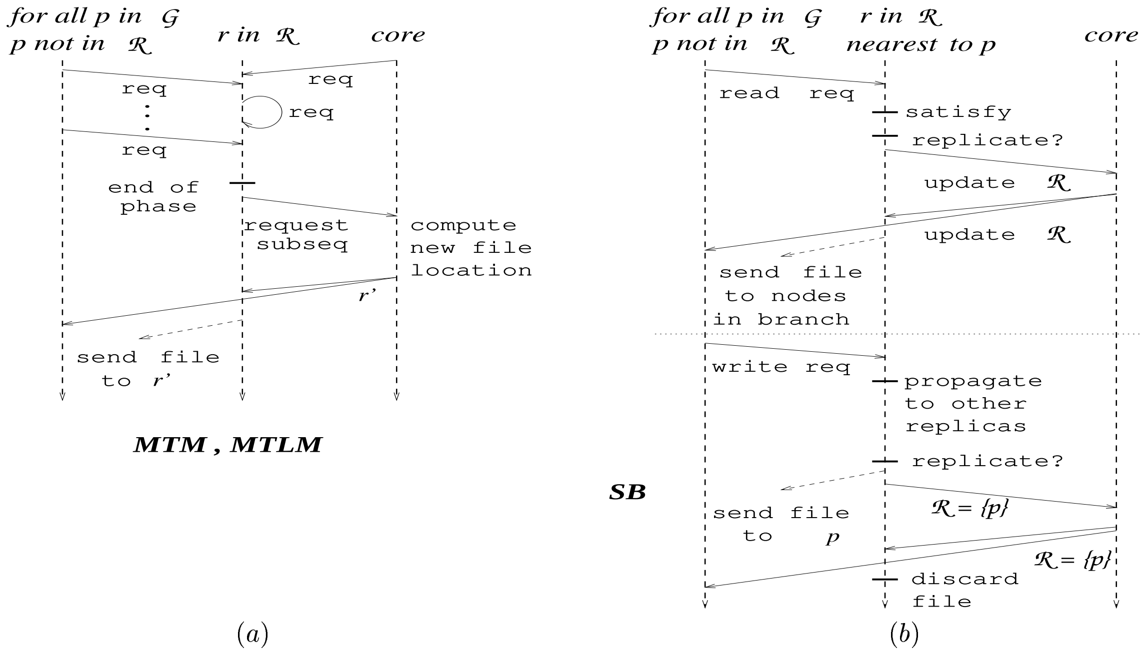



**Figure 2. Communication pattern for ($a$) the MTM and MTLM algorithms, and ($b$) the SB algorithm.**

Meanwhile, it records the processors issuing the requests in the current phase. At the end of the phase, $r$ sends the subsequence to the core. By exploiting its own topology awareness, the core computes the new file holder $r'$ by running the on-line algorithm, and it diffuses the $r'$'s identity along the tree. On receiving that message, each processor in $\mathcal{G}$ records $r'$ as the processor to which the requests must be addressed from that moment. On receiving that message, $r$ learns that the file must be moved to $r'$: it sends the file to $r'$ and then discards its own replica.

The communication pattern for the SB algorithm (figure 2($b$)) is similar. Each processor $p$ sends its own requests to the nearest replica $r$, which satisfies the request and then decides whether to change the replica allocation scheme, according to the probability functions described in section 3. If this is the case:

- for read requests, $r$ notifies the core of the $p$'s identity. The core runs the greedy ST algorithm [12] to join $p$, exploiting its own topology knowledge. Then, the core forwards along the tree a message with the identifiers of the nodes belonging to the new Steiner Tree branch. The replica nearest to each one of those processors sends to it a copy of the file;

- for write requests, the file is copied to $p$. Then, $r$ notifies the core that only one file replica must be maintained in $p$. The core forwards this message along the tree: all the processors currently in $\mathcal{R}$, different from $p$, discard their replica.

In both cases, the processors in $\mathcal{G}$ record the updated $\mathcal{R}$ membership. By their unicast routing tables, at every moment they can determine which is the nearest replica.

All the exchanged messages shown in figures 2($a$) and ($b$) are considered in our performance measures.

## 5. Performance evaluation

We implemented the MTM, MTLM and SB algorithms by using PARSEC (PARallel Simulation Environment for Complex systems) [13], a discrete-events, C-based simulation language. As the unicast routing protocol we implemented the *Distance Vector*. As the multicast routing protocol we adopted the *Core-Based Tree* (CBT) [4], that builds a bidirectional shared tree connecting the processors belonging to the group. We assume that the network is connected and that the links are reliable. All the links have weight 1; but, the network is not completely connected. The transmission time of a 1KB message along each link is 1 msec. The service time of a request at a processor is 1 msec. We generated a random network topology connecting 50 nodes. We considered different file sizes $F$, $\mathcal{G}$ membership cardinalities and request generation rates. For the SB algorithm, read requests are generated with probability $0.7$, while write requests are generated with probability $0.3$. To compare the MTM and MTLM algorithms, that proceed in phases, with the SB algorithm, we measured the performance of the latter one every $(0.7F + 0.3\alpha F) \simeq 1.22F$ requests, that is the expected frequency with which SB re-configures the file al-

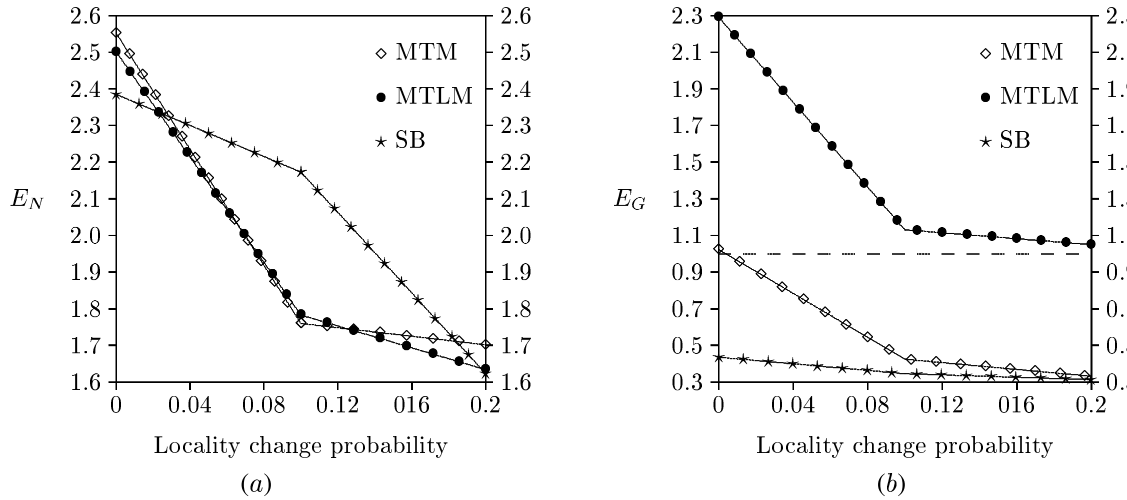



**Figure 3. ($a$) Net efficiency and ($b$) gross efficiency obtained by the three algorithms with respect to the probability of access locality changes per phase, with $|\mathcal{G}| = 10$ and $F = 5$.**

location.

We compare the performance achieved by the algorithms with that measured by never relocating the file (*static allocation policy*). As the performance indexes for an algorithm $\mathcal{A}$ we consider:

- the *net efficiency* $E_N(\sigma)$, defined as:

$$\frac{f_c(\sigma)\ with\ static\ allocation}{f_c(\sigma)\ with\ \mathcal{A}}$$

- the *gross efficiency* $E_G(\sigma)$, defined as:

$$\frac{f_c(\sigma)\ with\ static\ allocation}{(f_c(\sigma) + f_c(file\ relocation))\ with\ \mathcal{A}}$$

- the *latency time* with which a request is served.

$E_N$ measures the algorithm capability of moving the file close to the processors that more frequently access it. $E_G$ measures the gain in using the on-line algorithm instead of the static allocation policy: if $E_G < 1$ the static allocation is more convenient.

We initially evaluated the algorithm behaviours when group members are uniformily distributed over the network. The processors in $\mathcal{G}$ have different probabilities of generating requests, thus characterizing the access locality. We periodically change the access locality; we performed measures for different probability of changing locality in a phase[1]. The processors belonging to the new locality are randomly chosen using a $\beta$ distribution function. The measures have been averaged over 100 phases.

[1]For the SB algorithm, a "phase" lasts roughly $1.22F$ requests.

In figure 3 we report the $E_N$ and $E_G$ values for the studied algorithms with respect to the locality change rate. The MTM and MTLM algorithms have comparable $E_N$. They both efficiently characterize the access locality and relocate the file when the locality seldom changes. Otherwise, the request subsequence of the current phase does not provide an accurate prediction of the access locality of the next phase, and $E_N$ rapidly decreases. The shorter MTM phase duration with respect to MTLM makes it more prompt to adapt to locality changes. But, because of its promptness, MTM performs nearly 6 times more file migrations than MTLM for the same $\sigma$, characterizing the centre of gravity of the access locality by successive approximations, and as the consequence it is penalized in terms of $E_G$. When the locality never changes, MTLM places the file in the access locality centre with one migration; MTM performs 17 migrations. The SB algorithm tends to replicate the file on all the processors in the locality, because the read rate is higher than the write rate. When the locality changes, the processors in the new locality have to access the copies in the previous locality, as long as either: ($i$) new copies are created in the new locality, or ($ii$) a write request occurs that re-starts the ST subroutine by deleting all copies in the previous locality and creating one copy in the current locality. As the consequence, SB does not behave well in front of frequent access locality changes. Moreover, the continuous creation and deletion of file replicas negatively affects its $E_G$, whichever is the locality change probability.

These observations are confirmed by measuring $E_N$ and $E_G$ against $F$ (figure 4). The larger the file is, the more cautious all the algorithms are in changing the allocation scheme, to bound the management costs; hence, the file re-

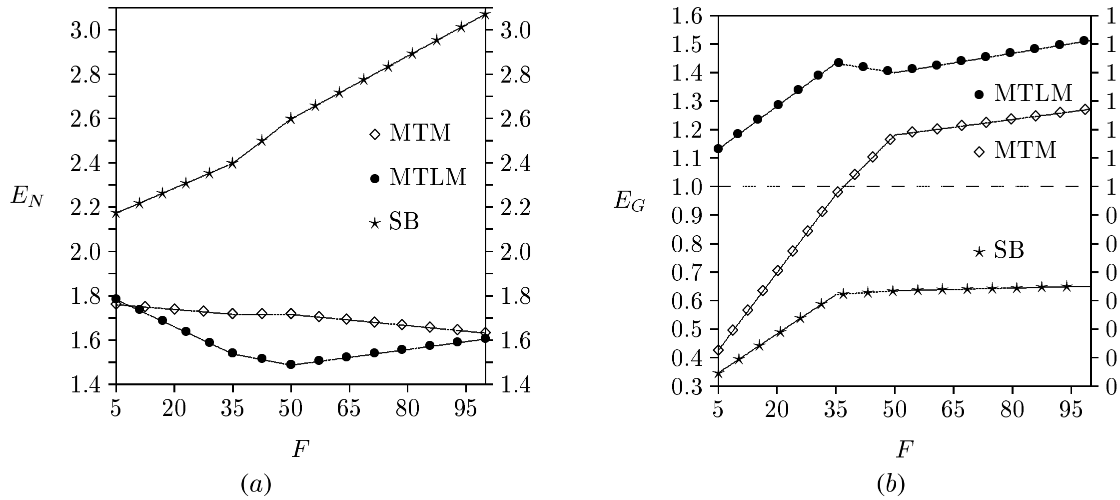



**Figure 4. ($a$) Net efficiency and ($b$) gross efficiency obtained by the three algorithms with respect to the file size $F$, with $|\mathcal{G}| = 10$ and locality change probability $0.1$.**

locations halve passing from $F = 5$ to $F = 100$ and $E_G$ increases. MTLM shows a more stable behaviour than MTM and SB, because it always considers the migration costs in deciding the file relocation. The SB algorithm changes the file allocation scheme with an expected rate roughly equal to that of MTM. As noticed in section 2.1, the availability of several replicas, placed at the processors in $\mathcal{G}$, and the low write probability, allow SB to achieve high efficiency in satisfying the access requests. By contrast, MTM and, in particular, MTLM become slower in adapting to the new locality as $F$ increases (figure 4($a$)).

The increase in $|\mathcal{G}|$ decreases both $E_N$ and $E_G$ for all the algorithms, because the requesting processors are scattered in the system and their centre of gravity is more difficult to characterize. The relationships among the algorithms are the same as in figure 4.

We also evaluated the performance in the presence of *hot spots*. We set $|\mathcal{G}| = 25$ and we characterized 5 subsets of $\mathcal{G}$, each one composed of 5 near nodes, located in different network regions. Each subset cyclically becomes the access locality for 10 phases. Under these conditions, both MTM and MTLM have $E_G$ around 4, while SB is on average below 1.5. In figure 5($a$) we show $E_N$ and $E_G$ for SB against time; MTM and MTLM show comparable behaviours. By contrast with the uniform case, both efficiencies highly improve because the access locality is always topologically well defined. The efficiency drops correspond with locality changes, and they are quickly overcome by the MTM and MTLM algorithms. By contrast, SB is slower in recovering by the same arguments discussed with regard to figure 3. On the other hand, SB is able to efficiently distribute the request load among several replicas, differently from MTM and MTLM, thus guaranteeing low response times to the requesting processors (figure 5($b$)).

## 6. Concluding remarks

In this work we study the policies to dynamically relocate shared files in a distributed system, with the aim of minimizing the access and management costs. As the allocation policies, we consider those adopted by the algorithms proposed in the literature to solve the on-line file migration and file allocation problems. We devised an implementation scheme for those algorithms. We implemented some of them, and we evaluated their performance in realistic settings.

All the algorithms allow to reduce the access costs with respect to the static allocation policy, sometimes at the expenses of the management costs. The static allocation policy is only convenient when the access locality does not change over time, and can be determined a-priori. In this case, the shared file should be statically placed in the centre of gravity of the subnetwork connecting the file users. In the other cases, a dynamic allocation policy is preferable. Policies maintaining several copies of the file (such as SB) minimize both the access cost and the latency in serving the requests, by distributing the request load on different servers placed in the users neighbourhood. Those policies are particularily convenient in the case the most part of the requests are read requests, rather than write requests. On the other hand, the maintenance of several copies has high management costs.

By contrast, policies maintaining only one file copy are advantageous when the file content is frequently updated.

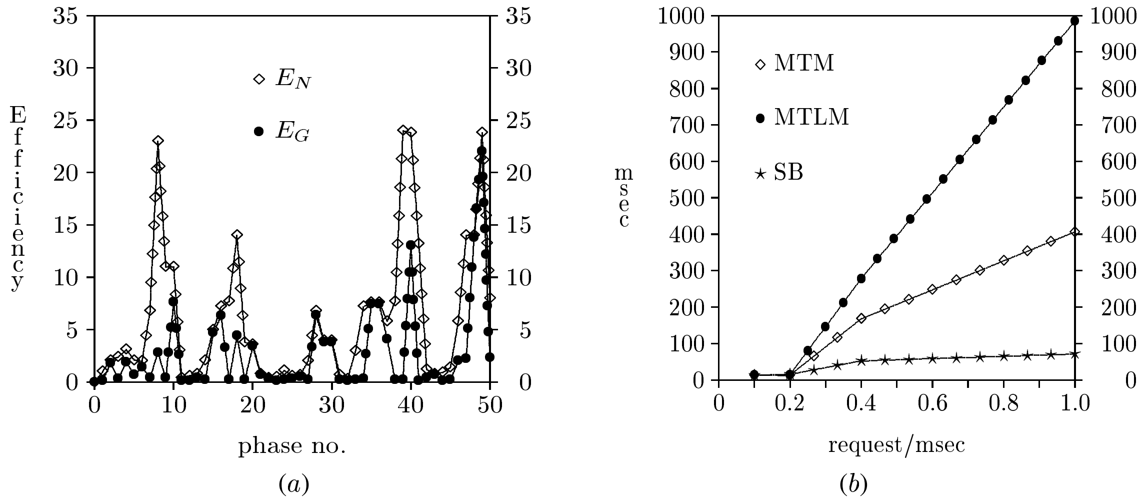



**Figure 5. ($a$) Net and gross efficiency with respect to the time for the SB algorithm with $F = 5$ and hot spot request generation. ($b$) Latency time for the three algorithms with respect to the request load, with $F = 35$ and hot spot request generation.**

They provide their best performance when either they take into account the management costs in relocating the file (as in the case of the MTLM algorithm), or in the case of large file size. However, the unique server may become a bottleneck, thus penalizing the request service time.

The on-line algorithms have been designed to achieve good performance in the worst case. We are currently trying to improve their behaviours in the average case, by intervening on both $(i)$ the probability function that drives the allocation scheme changes, and $(ii)$ the technique used to characterize the new scheme.

## References


[1] Ajtai M., Aspnes J., Dwork C., Waarts O. "A Theory of Competitive Analysis for Distributed Algorithms". *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 401–411, Nov. 1994.

[2] Awerbuch B., Bartal Y., Fiat A. "Competitive Distributed File Allocation". *Proc. 25th ACM STOC*, pages 164–173, 1993.

[3] Awerbuch B., Bartal Y., Fiat A. "Heat & Dump: Competitive Distributed Paging". *Proc. 34th IEEE Annual Symp. on Foundations of Computer Science*, pages 22–31, Nov. 1993.

[4] Ballardie T., Francis P., Crowcroft J. "Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing". *Proc. SIGCOMM'93*, pages 85–94, Sep. 1993.

[5] Bartal Y. "Survey on Distributed Paging". *Proc. Dagstuhl Workshop on On-line Algorithms*, Jul. 1996.

[6] Bartal Y., Charikar M., Indyk P. "On Page Migration and Other Relaxed Task Systems". *Proc. 8th Ann. ACM-SIAM Symp. on Discrete Algorithms*, Jan. 1997.

[7] Bartal Y., Fiat A., Rabani Y. "Competitive Algorithms for Distributed Data Management". *Journal of Computer and System Sciences, Vol. 51*, pages 341–358, 1995.

[8] Ben-David S., Borodin A., Karp R., Tardos G., Wigderson A. "On the Power of Randomization in On-line Algorithms". *Algorithmica, Vol. 11, No. 1*, pages 2–14, 1994.

[9] Black D. L., Sleator D. D. "Competitive Algorithms for Replication and Migration Problems". *Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University*, 1989.

[10] Estrin D. et al. "Protocol Independent Multicast - Sparse Mode (PIM-SM)". *RFC2362*, Jun. 1998. Work in progress.

[11] Fiat A., Woeginger G. (editors). "On-line Algorithms". *Dagstuhl Seminar Report; 149*, Jun. 1996.

[12] Imaze M., Waxman B. M. "Dynamic Steiner Tree Problem". *SIAM Journal on Discrete Mathematics, Vol. 4, No. 3*, pages 369–384, Aug. 1991.

[13] Meyer R. A. "PARSEC User Manual - Release 1.1". *UCLA Parallel Computing Laboratory*, Aug. 1998.

[14] Sleator D. D., Tarjan E. "Amortized Efficiency of List Update and Paging Rules". *Communications of the ACM, Vol. 28, No. 2*, pages 202–208, 1985.