IEICE TRANS. INF. & SYST., VOL.E86–D, NO.12 DECEMBER 2003

PAPER

Comparative Performance Analysis of Ordering Strategies in Atomic Broadcast Algorithms

Xavier DÉFAGO[†,††], ***Regular Member***, **André SCHIPER**[†††], ***and*** **Péter URBÁN**[†††], ***Nonmembers***

SUMMARY In this paper, we present the results of a comparative analysis of Atomic Broadcast algorithms. The analysis was done by using an analytical method to compare the performance of five different classes of Atomic Broadcast algorithms. The five classes of Atomic Broadcast algorithms are determined by the mechanisms used by the algorithms to define the delivery order. To evaluate the performance of algorithms, the analysis relies on contention-aware metrics to provide a measure for both their latency and their throughput. The results thus obtained yield interesting insight into the performance tradeoffs of different Atomic Broadcast algorithms, thus providing helpful information to algorithms and systems designers.

key words: *distributed algorithms, Atomic Broadcast, total order, performance analysis, contention-aware metrics*

1. Introduction

Atomic Broadcast, also sometimes called Total Order Broadcast, is a fundamental problem for distributed systems. Informally, the problem is defined as a broadcast primitive whereby all processes deliver the same sequence of messages. For instance, it is a powerful abstraction for solving problems such as process replication (using the state machine approach) [15] and support for transactions in replicated databases [12].

There exists a vast amount of literature about Atomic Broadcast, with more than sixty different algorithms published [8]. However, important as it is, there have been few analyses on the performance tradeoffs between those algorithms. Cristian et al. [6] study four different algorithms and compare them using discrete event simulation with random communication delays. Friedman and van Renesse [9] measure the performance of six different algorithms over a local network. They confirm the general observations made by the former study, and show that packing messages can significantly improve the performance of Atomic Broadcast algorithms. These two analyses are indeed interesting, but they are limited to a few classes of algorithms.

The aim of our study is to highlight the performance tradeoffs inherent to each algorithm, thus helping system designers in their choice. In this paper, we provide a *quantitative* comparison of Atomic Broadcast algorithms, which builds upon previous work on a *qualitative* comparison of those algorithms [8]. We measure the performance of algorithms using contention-aware metrics [16],* which evaluate both the *latency* and the *throughput* of algorithms. Unlike traditional complexity metrics, those metrics take account of contention on two types of resources: CPU and network.

The efforts needed to compute the metrics for every single algorithm published so far are of course totally disproportionate, and would anyway ask for a much more precise description of the algorithms. To overcome this problem, we use the classification system defined when surveying of more than sixty different Atomic Broadcast algorithms [8]. We rely on the observation that two different algorithms which belong to the same class generally have similar communication patterns. Thus, we can compare *classes* rather than individual algorithms, unlike what was done in previous analyses. Consequently, our results are more general in scope, but somewhat less accurate. Beside, we can *compare* classes of algorithms, but not obtain a realistic estimation of their absolute performance. The results obtained are however significantly more accurate than when using only time and message complexity, and more general than those obtained with simulation. In short, we use an analytic approach, in contrast to Cristian et al. [6] who use simulation, and Friedman and van Renesse [9] who measure the actual performance of the algorithms in a specific settings.

The aim of our study is to highlight important performance tradeoffs associated with the main ordering strategies. More concretely, our results show that the choice of an appropriate algorithm depends greatly on the requirements of the system in which it is to be used. For instance, we confirm that communication history algorithms usually provide a good throughput at the expense of latency, and that fixed sequencer algorithms have good latency but poor throughput. In most cases,

Manuscript received January 20, 2003.
Manuscript revised June 10, 2003.

[†]The author is with the School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa-ken, 923–1292 Japan.
[††]The author is with PRESTO, Japan Science and Technology Agency, Saitama-ken, 332–0012 Japan.
[†††]The authors are with the Faculty of Information and Communication Sciences, Swiss Federal Institute of Technology in Lausanne (EPFL), 1015 Lausanne, Switzerland.

*In [16] we also present a comparison of Atomic Broadcast algorithms as an illustration for the metrics. However, that previous analysis is limited to a few algorithms.

moving sequencer algorithms provide a comprise. We also show that, contrary to popular belief, the ratio between computer and network speed is also a determinant factor when choosing an ordering strategy.

The rest of the paper is structured as follows. Section 2 presents the system model and important definitions. Section 3 defines the representative algorithms. Section 4 presents the contention-aware metrics. Section 5 analyzes the non-uniform algorithms, followed by Sect. 6 for the uniform ones. Section 7 consolidates the observations made previously and concludes the paper.

2. System Model and Definitions

The algorithms considered in this paper are based on an asynchronous system model (i.e., no assumption on messages delays or process speed). Communication channels are reliable and FIFO. Processes can only fail by crashing, and failures are handled by a group membership service. The group membership is left unspecified because we do not analyze the case when failures (or suspicions) occur. If we did, we would end up comparing different group membership implementations instead of Atomic Broadcast algorithms.

2.1 Definition of Atomic Broadcast

Formally, Atomic Broadcast is defined in terms of two primitives called *A-broadcast*(m) and *A-deliver*(m), where m is some message. When a process p executes *A-broadcast*(m) (resp., *A-deliver*(m)), we may say that p A-broadcasts m (resp., A-delivers m). We assume that every message m can be uniquely identified, and carries the identifier of its sender, given by *sender*(m). In this context, Atomic Broadcast is defined by the following properties [4], [10]:

(Validity) If a correct process A-broadcasts a message m, then it eventually A-delivers m.

(Agreement) If a correct process A-delivers a message m, then all correct processes eventually A-deliver m.

(Integrity) For any message m, every process A-delivers m at most once, and only if m was previously A-broadcast by *sender*(m).

(Total Order) If correct processes p and q both A-deliver messages m and m', then p A-delivers m before m' if and only if q A-delivers m before m'.

In this paper, we also consider the stronger definition of *uniform* Atomic Broadcast which is defined by stronger properties for agreement and total order.

(Uniform Agreement) If a process *(correct or not)* A-delivers a message m, then all correct processes eventually A-deliver m.

(Uniform Total Order) If processes p and q *(correct or not)* both A-deliver messages m and m', then p A-delivers m before m' if and only if q A-delivers m before m'.

Uniform properties are required by certain classes of applications, such as atomic commit or active replication. For instance, in systems where replicated processes can have side-effects, using a non-uniform Atomic Broadcast algorithm could result in conflicting actions from different replicas. However, uniformity is not required by all applications, and unfortunately has a significant cost. For this reason, we consider here both *uniform* and *non-uniform* algorithms, but separately.

3. Atomic Broadcast Algorithms

In this analysis, we consider algorithms to represent each of five classes of Atomic Broadcast algorithms defined in previous work [8]. The classification, based on the ordering mechanism, was determined by surveying around 60 algorithms. It relies on the fact that the ordering is generated by either one of three types of processes: *sender*, *destinations*, or a *sequencer*. With other factors, this leads to the following five classes: *fixed sequencer*, *moving sequencer*, *privilege-based*, *communication history*, and *destinations agreement*.

In this section, we describe each class and its representative algorithms. The latter are just simplifications of existing algorithms. Whenever possible, we consider both a uniform and a non-uniform variant. For readability and conciseness, we describe each algorithm informally, illustrating their execution on a time-space diagram (see Table 1). Each scenario shows an execution of the algorithm wherein a single process broadcasts one message m. The communication pattern thus described is sufficient for computing the metrics.

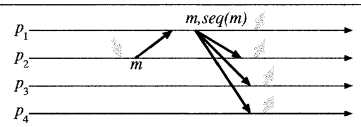
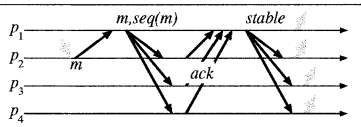
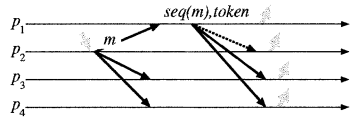
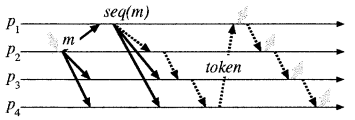
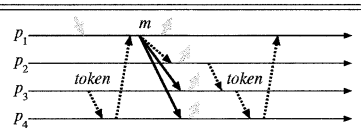
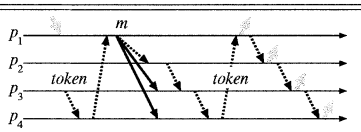
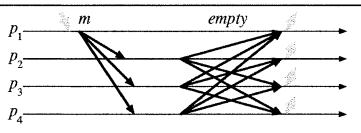
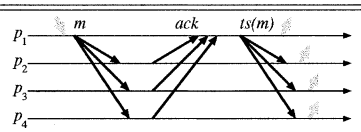
3.1 Fixed Sequencer

Fixed sequencer algorithms are by far the simplest. The idea is that one process in the system is elected as a sequencer that is responsible for ordering all messages.

The representative algorithms run as follows (see Table 1): when a process p wants to broadcast a message m, it sends m to the sequencer. The sequencer assigns a sequence number to m, and sends both m and the sequence number to the other processes. In the non-uniform algorithm, processes deliver m as soon as they receive it with its sequence number. In the uniform algorithm, the processes can deliver m only after it has been acknowledged by all processes.

Fixed sequencer algorithms are rarely uniform. This is probably because uniformity is comparatively more expensive than for algorithms of other classes. Isis [3] and Amoeba [11] are two well-known examples of fixed sequencer algorithms.

Table 1 Representative uniform and non-uniform algorithms for each class.

	Non-uniform	Uniform
Fixed sequencer		
Moving sequencer		
Privilege-based		
Communication history	*no algorithm*	
Destinations agreement		*no algorithm*

3.2 Moving Sequencer

With moving sequencer algorithms, the role of sequencer is passed from one process to another. This is done by a token which carries a sequence number and constantly circulates among the processes.

For both the uniform and non-uniform algorithms, Table 1 illustrates a run in which process p_2 broadcasts a message m and process p_1 is the current token holder (i.e., sequencer). The messages that carry the token are represented by a dashed arrow.

In short, the non-uniform algorithm works as follows. When a process p wants to broadcast a message m, it sends m to all other processes. Upon receiving m, processes store it into a receive queue. When the current token holder q has a message in its receive queue, it assigns a sequence number to the first message in the queue and broadcasts that number together with the token.[†] A process can then deliver m when it has (1) received m, (2) received its sequence number, and (3) delivered every message which comes before m.

The uniform algorithm is similar, except that the destination processes must also wait until m is stable (i.e., until it has been acknowledged by all processes) before they can deliver it. The detection of stability is performed by the token which gathers the acknowledgments. Besides, in the moving sequencer algorithms, the token does not need to rotate all the time, and thus eventually stops. In the non-uniform algorithm, the token can stop earlier than with its uniform counterpart because the detection of stability is not necessary.

The Atomic Broadcast algorithm proposed by Chang and Maxemchuck [5] is a well-known example of a moving sequencer algorithm.

3.3 Privilege-Based

With privilege-based algorithms, the delivery order is determined by the senders. When a process has a message to broadcast, it must first obtain that privilege.

The non-uniform algorithm works as follows (see Table 1). When a process p wants to broadcast a message m, it simply stores m into a send queue until it receives the token. The token carries a sequence number and constantly circulates amongst the processes. When p receives the token, it extracts m from its send queue, uses the sequence number carried by the token, and broadcasts m with the sequence number. Then, p increments the sequence number and transmits the token to the next process. To reduce the number of messages, p can broadcast the token along with m in a single message. When a process receives m, it delivers m according to its sequence number.

In the uniform algorithm (Table 1), the token also carries the acknowledgments. Before delivering a message m, processes must wait until m is stable, which requires a full round-trip of the token.

Totem [1] is a typical illustration of a privilege-based Atomic Broadcast algorithm. A less typical example is the on-demand protocol [7].

[†]The token carries the identity of the next token holder. To reduce traffic, the token is embedded in the broadcast message, and ignored by all but the next token holder.

3.4 Communication History

With communication history algorithms, the delivery order is determined by the senders, just like with privilege-based algorithms. The main difference is that processes can send messages at any time. The destinations observe the messages generated by the other processes to learn when delivering a message will no longer violate the total order. In most cases, communication history algorithms use a partial order defined by the causal history of messages and transform this partial into a total order, where concurrent messages are ordered according to some predetermined function.

The communication history algorithm considered in this paper (see Table 1) works as follows. A partial order is generated by using logical clocks [13] to "timestamp" each message m with the logical time of the *A-broadcast*(m) events. This partial order is then transformed into a total order by using the lexicographical order on the identifiers of sending processes to arbitrate ties: if two messages m and m' have the same logical timestamp, then m is before m' if $id(sender(m)) < id(sender(m'))$, where $id(p)$ is the identifier of process p. It follows that a process p can deliver some message m only once it knows that no message m' received in the future will carry a lower timestamp (or an equal timestamp with $id(sender(m')) < id(sender(m))$).

The algorithm is not live as it stands, because a silent process could prevent other processes from delivering. To avoid this, a process p is required to broadcast an *empty* message after a delay Δ_{live}, if it has nothing else to broadcast. When computing the latency metrics, we make the simplifying assumption that $\Delta_{\text{live}} = 0$. In the case of computing the throughput, the scenario does not generate any empty message anyway.

Communication history algorithms are an application of Lamport's mutual exclusion algorithm based on logical clocks [13]. Psync [14] is well-known example.

3.5 Destinations Agreement

With destinations agreement algorithms, the delivery order is determined by the destination processes. In short, this is done in one of two ways; (1) ordering information generated by every process is combined deterministically, or (2) the order is obtained by an agreement between the destinations (e.g., consensus).

The algorithm considered in this paper uses the first approach and is adapted from Skeen's algorithm (as described in [2]). The algorithm works as follows (see Table 1). To broadcast a message m, process p_1 sends m to all processes and acts as a coordinator for the delivery of m. Upon receiving m, a process q sends an acknowledgment and a logical timestamp $ts_q(m)$ back to p. Process p_1 gathers all timestamps and computes the final timestamp $TS(m)$ as the maximum of all received timestamps. Finally, p_1 sends $TS(m)$ to all processes which deliver m according to $TS(m)$.

An example of the second approach, based on consensus, is due to Chandra and Toueg [4].

4. Contention-Aware Metrics

In this paper, we compare Atomic Broadcast algorithms using the contention-aware metrics defined in [16]. We briefly describe but do not motivate them, as this is already done extensively in [16].

The contention-aware metrics evaluate distributed algorithms according to their *latency* or to their *throughput*. The two metrics take account of resource contention that occurs for both *CPU* and *network*. The definitions of the metrics are based on a common system model, denoted $\mathcal{M}_{\text{pp}}(n, \lambda)$, which assumes a point-to-point network. Finally, we define a variant of the metrics based on a second model ($\mathcal{M}_{\text{br}}(n, \lambda)$) which supports broadcast communication.

4.1 Point-to-Point Model

The model is defined around a single network resource and one CPU resource attached to each process. When a message m is transmitted from a process p_i to a destination process p_j, m must consecutively acquire and release the following resources: CPU_i, network, and CPU_j (see Fig. 1). Resources are accessed in mutual exclusion, and a fixed cost is associated with each one. The transmission of m from p_i to p_j occurs as follows:

1. m enters the *sending queue*† of p_i, waiting for CPU_i to be available.
2. m takes the resource CPU_i for λ time units, where λ is a parameter of the system model ($\lambda \in \mathbb{R}_0^+$).
3. m enters the *network queue* of p_i and waits until the network is available for transmission.
4. m takes the network resource for 1 time unit.
5. m enters the *receiving queue* of p_j and waits until CPU_j is available.
6. m takes the resource CPU_j of p_j for λ time units.

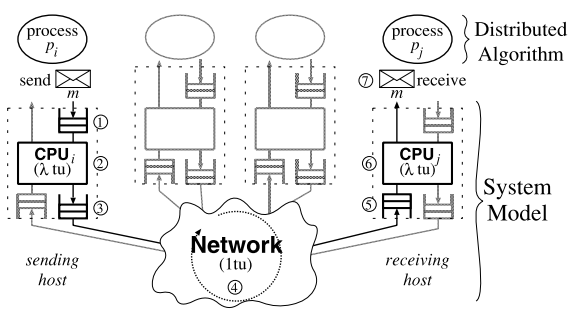


Fig. 1 Decomposition of the end-to-end delay (tu=time unit).

†All queues (sending, receiving, and network queues) in the model use a FIFO policy.

7. Message m is received by p_j in the algorithm.

Resource conflicts are resolved according to the following rules, thus making the model *deterministic*.

Network. Concurrent requests to the network may arise when messages at different hosts are simultaneously ready for transmission. The access to the network is modeled by a round-robin policy.

CPU. For each CPU resource, conflicts between outgoing messages (i.e., in sending queue) and incoming ones are resolved by giving the priority to the outgoing messages.

Send to all (point-to-point). If p is a process that sends a message m to all processes, then p sends the message m consecutively to all processes in the lexicographical order $(p_1, p_2, \ldots, p_n)$ except itself.

Definition 1 (point-to-point): Model $\mathcal{M}_{\mathrm{pp}}(n, \lambda)$ is the model described above, with parameters $n \in \mathbb{N}$ and $\lambda \in \mathbb{R}_0^+$, where $n > 1$ is the number of processes and λ is the relative cost between CPU and network.

4.1.1 Latency Metric

The definition of the latency metric uses the terms: "start" and "end" of a distributed algorithm. The definition of these terms depends on the problem that an algorithm $\mathcal{A}$ solves. In the case of Atomic Broadcast, one considers the broadcast of a single message m. Then, "start" is the execution of $A\text{-}broadcast(m)$ and "end" is the execution of the last $A\text{-}deliver(m)$.

Definition 2 (latency, point-to-point): Let $\mathcal{A}$ be a distributed algorithm. The metric $\mathrm{Latency}_{\mathrm{pp}}(\mathcal{A})(n, \lambda)$ is defined as the number of time units between the start and the end of algorithm $\mathcal{A}$ in model $\mathcal{M}_{\mathrm{pp}}(n, \lambda)$.

4.1.2 Throughput Metric

The throughput metric of an algorithm $\mathcal{A}$ relates to the usage of system resources in one run of $\mathcal{A}$. The most heavily-used resource acts as a bottleneck and sets a limit on the *maximal throughput*; an upper bound on the frequency at which the algorithm can be run.

Definition 3 (throughput, point-to-point): Let $\mathcal{A}$ be a distributed algorithm. The throughput metric is defined as follows:

$$\mathrm{Thput}_{\mathrm{pp}}(\mathcal{A})(n, \lambda) \stackrel{\mathrm{def}}{=} \frac{1}{\max_{r \in \mathcal{R}_n} T_r(n, \lambda)} \qquad (1)$$

where $\mathcal{R}_n$ denotes the set of all resources (i.e., the network and $\mathrm{CPU}_1, \ldots, \mathrm{CPU}_n$), and $T_r(n, \lambda)$ denotes the total duration for which resource $r \in \mathcal{R}_n$ is utilized in one run of algorithm $\mathcal{A}$ in model $\mathcal{M}_{\mathrm{pp}}(n, \lambda)$.

Notice that the definition above (Def. 3) is in fact a simplification of the actual one. When there are several possible execution patterns for the same algorithm, $T_r(n, \lambda)$ becomes the duration *amortized* over all possible cases and their occurrence rate. For instance, with a moving sequencer algorithm, each process is sequencer $1/n$-th of the time, and a normal process otherwise. All formulas computed in this paper use amortized cost.

4.2 Broadcast Model

The definition of the broadcast model $\mathcal{M}_{\mathrm{br}}(n, \lambda)$ is similar to that of the point-to-point model, except for "send to all" which is replaced by the following rule:

Send to all (broadcast). If p is a process that sends a message m to all, then p sends a single copy of m, the network transmits a single copy of m, and each process (except p) receives a copy of m.

The definition of $\mathrm{Latency}_{\mathrm{br}}(\mathcal{A})(n, \lambda)$ differs from that of $\mathrm{Latency}_{\mathrm{pp}}(\mathcal{A})(n, \lambda)$ (resp., $\mathrm{Thput}_{\mathrm{br}}(\mathcal{A})(n, \lambda)$ differs from $\mathrm{Thput}_{\mathrm{pp}}(\mathcal{A})(n, \lambda)$) only by the fact that the former is defined in broadcast model $\mathcal{M}_{\mathrm{br}}(n, \lambda)$ instead of point-to-point model $\mathcal{M}_{\mathrm{pp}}(n, \lambda)$.

5. Evaluation of Non-Uniform Algorithms

We use the contention-aware metrics to analyze the latency of the non-uniform algorithms. We discuss the four non-uniform algorithms: *non-uniform privilege-based*, *non-uniform moving sequencer*, *non-uniform fixed sequencer*, and *destinations agreement*. For the sake of readability, we only give a graphical representation of the results obtained with the metric. The exact formulas are given in the appendix.

5.1 Point-to-Point Networks

5.1.1 $\mathrm{Latency}_{\mathrm{pp}}$(Non-Unif. Atomic Broadcast)(n, λ)

Figure 2 shows the latency of the non-uniform algorithms in point-to-point networks. The figure has three parts: the top gives a three-dimensional view of the latency, according to the two parameters n and λ; the lower left part shows the curves when λ is fixed ($\lambda = 1$) and n varies; similarly, the lower right part shows the curves when n is fixed ($n = 10$) and λ varies.

(best) fixed seq. $<$ moving seq.
$<$ privilege $<$ dest. agr. (worst)

Figure 2 shows that the four algorithms form two groups, with a clear separation between the moving sequencer and the privilege-based algorithms. In addition, the fixed and moving sequencer algorithms have the same latency (see Fig. 2, lower right) for large values of λ. This last point is explained by the lesser impact of network contention.

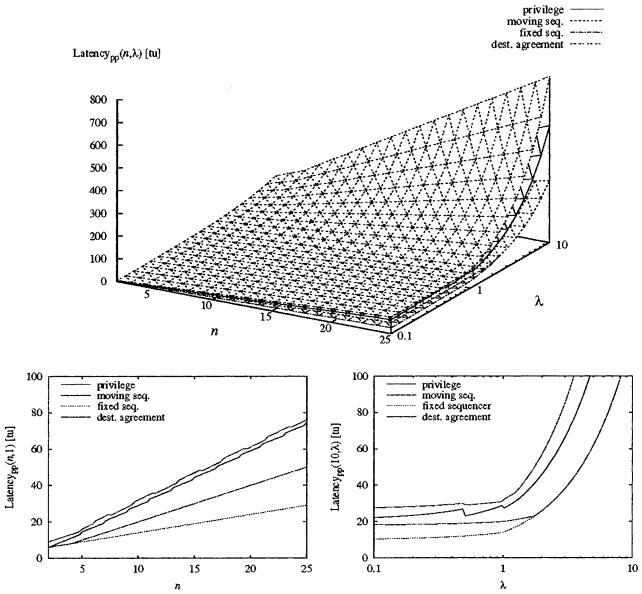


Fig. 2 Graphical representation of $\text{Latency}_{\text{pp}}(\mathcal{A})(n, \lambda)$ for non-uniform algorithms.

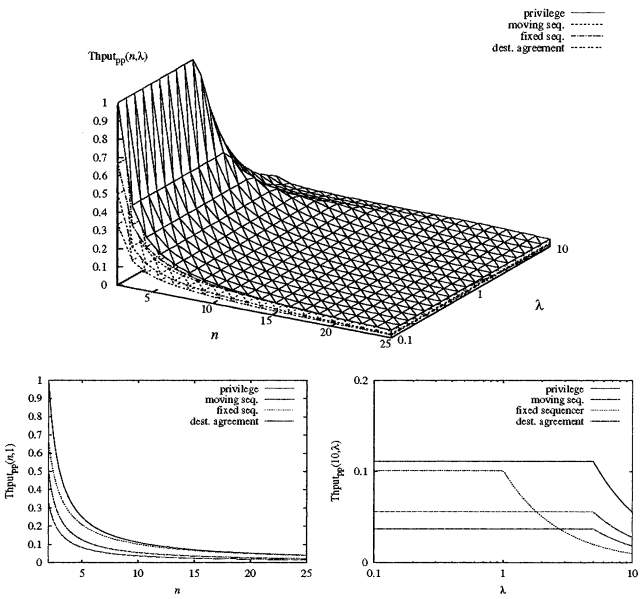


Fig. 3 Graphical representation of $\text{Thput}_{\text{pp}}(\mathcal{A})(n, \lambda)$ for non-uniform algorithms.

5.1.2 Thput_{pp}(Non-Unif. Atomic Broadcast)(n, λ)

Figure 3 shows the throughput of the non-uniform algorithms. In addition, Fig. 4 gives a detailed view of the upper part of Fig. 3, where the privilege-based and the destinations agreement algorithms have been removed to make visible the intersection between the fixed and the moving sequencer algorithms.

(best) privilege > moving seq. > dest. agreement

The privilege-based algorithm has clearly the best

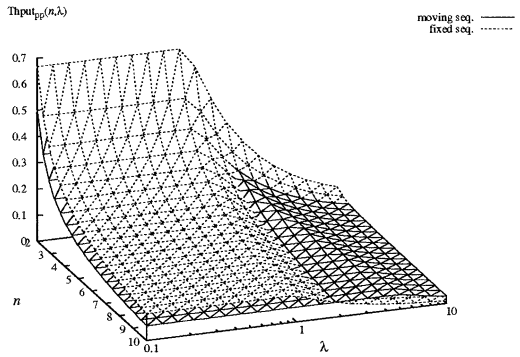


Fig. 4 Graphical representation of non-uniform moving and fixed sequencer algorithms (detail of Fig. 3).

throughput (see Fig. 3). In spite of their differences, the privilege-based, the moving sequencer, and the destinations agreement algorithms have a comparable behavior with respect to parameter changes.

The most interesting point is the behavior of the fixed sequencer algorithm. Indeed, its performance largely depends on the value of λ: the algorithm moves from the second position when λ is small, to the last position when λ is big. This is visible in Fig. 3 (lower right) and on Fig. 4. The fixed sequencer has a good throughput when λ is small. However, the sequencer becomes a bottleneck when λ increases. The moving sequencer and the destinations agreement algorithms are less prone to this problem as they better distribute the load among the processes.

5.2 Broadcast Networks

5.2.1 $\text{Latency}_{\text{br}}$(Non-Unif. Atomic Broadcast)(n, λ)

Figure 5 shows the latency for the non-uniform algorithms in a broadcast network. In a broadcast network, the latency of the moving sequencer and the fixed sequencer algorithms are equal. For this reason, both algorithms are plotted together.

It is clear from Fig. 5 that the sequencer algorithms (fixed and moving) have the best latency. The extra overhead that the moving sequencer algorithm has to pay for the first broadcast in a point-to-point network, disappears in a broadcast network. As a result, both sequencer algorithms have the same latency.

Figure 5 shows an interesting comparison between privilege-based and destinations agreement algorithms. First, Fig. 5 (lower left) clearly shows that the latter has the best latency when n is large. But, the relative performance of the algorithms when λ varies is even more interesting, as shown in Fig. 5 (top; lower right). The privilege-based algorithm performs better than the destinations agreement for small values of λ, but also for large values of λ. This is because the destinations agreement algorithm benefits from a concurrent use of both network and CPU resources.

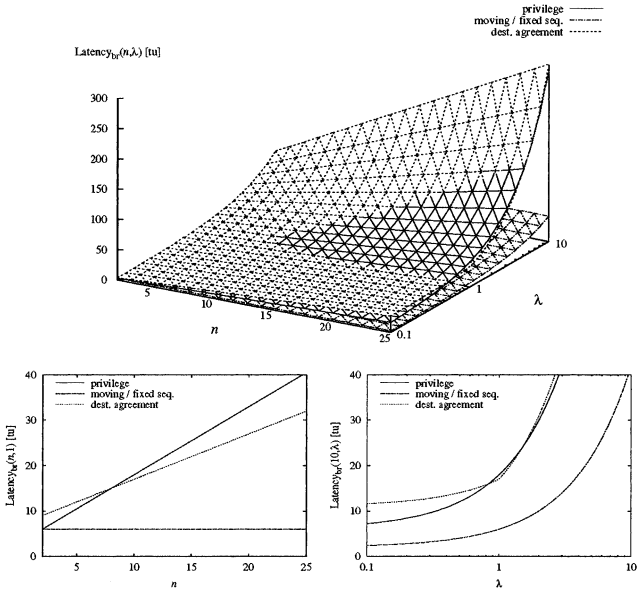


Fig. 5 Graphical representation of $\text{Latency}_{\text{br}}(\mathcal{A})(n, \lambda)$ for non-uniform algorithms.

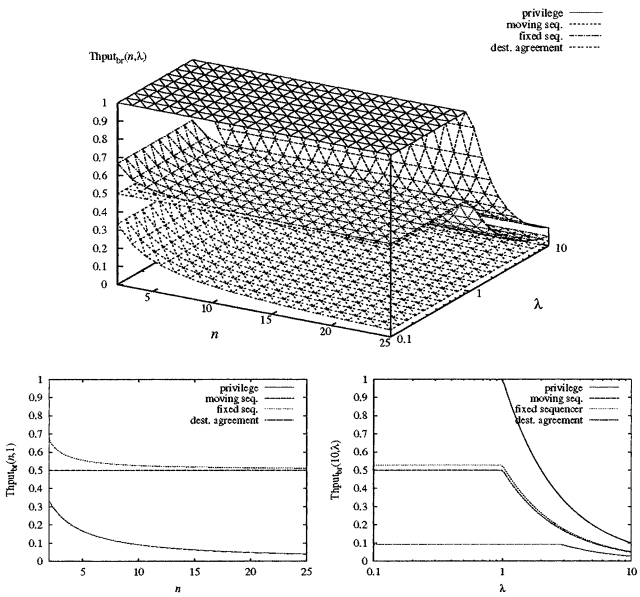


Fig. 6 Graphical representation of $\text{Thput}_{\text{br}}(\mathcal{A})(n, \lambda)$ for non-uniform algorithms.

5.2.2 Thput_{br}(Non-Unif. Atomic Broadcast)(n, λ)

Figure 6 depicts the throughput of the non-uniform algorithms in a broadcast network. The algorithms rank as follows.

(best) privilege > fixed seq.
> moving seq. > dest. agr. (worst)

The two sequencer algorithms (moving and fixed) perform similarly, especially when n grows (see Fig. 6; lower left). The throughput of the fixed sequencer decreases when n increases, because of the first message sent by the algorithm. Indeed, in the fixed sequencer algorithm, the algorithm does not send the first message when the sender happens to be the sequencer. This situation reduces resource usage, but occurs less frequently as n increases. This explains why the fixed sequencer algorithm asymptotically (in terms of n) behaves like the moving sequencer algorithm.

Figure 6 (lower left) leads to a second observation. Unlike the other three algorithms, the destinations agreement algorithm asymptotically tends to a null throughput as n grows, despite the broadcast network. This lack of scalability is due to the local timestamps that the sender has to gather from all destination processes. Broadcast network or not, this generates $O(n)$ messages.

On the positive side, the destinations agreement algorithm is less sensitive to large values of λ than the other algorithms. This is visible in Fig. 6 (lower right), where the throughput starts to drop when λ is larger than 3 (instead of $\lambda > 1$ for the other algorithms).

6. Evaluation of Uniform Algorithms

We analyze the latency of the uniform algorithms: *communication history*, *uniform privilege-based*, *uniform moving sequencer*, and *uniform fixed sequencer*. We first consider these algorithms in a point-to-point network, and illustrate the results of both $\text{Latency}_{\text{pp}}(\mathcal{A})(n, \lambda)$ and $\text{Thput}_{\text{pp}}(\mathcal{A})(n, \lambda)$ in Figs. 7 and 8, respectively. Then, we analyze the same algorithms in a broadcast network using $\text{Latency}_{\text{br}}(\mathcal{A})(n, \lambda)$ (Fig. 9) and $\text{Thput}_{\text{br}}(\mathcal{A})(n, \lambda)$ (Fig. 10). The exact formulas are given in the appendix.

6.1 Point-to-Point Networks

6.1.1 $\text{Latency}_{\text{pp}}$(Uniform Atomic Broadcast)(n, λ)

Figure 7 represents the latency of the four uniform algorithms according to the two parameters: n and λ. According to conventional complexity metrics, the fixed sequencer and the communication history algorithms have the best latency (latency degree of 2). Then, the moving sequencer and privilege-based algorithms both have a worse latency. This conclusion seems simplistic when compared to the results obtained on Fig. 7.

As a first observation, Fig. 7 (lower left) shows that the quadratic number of messages generated by the communication history algorithm causes a lot of network contention, and has a strong influence on the latency. For instance, because of this contention, the communication history algorithm has the worst latency in a system with more than 10 processes. In contrast, this algorithm has the best latency in a system with less than five processes. This is easily explained by the

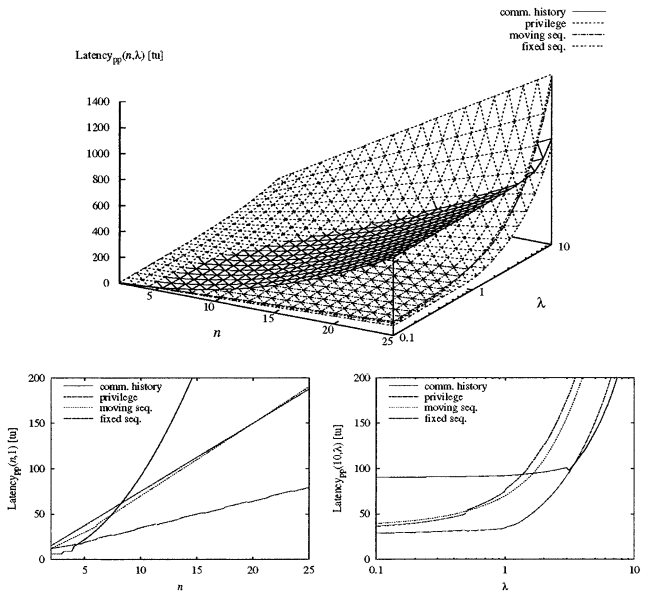


Fig. 7 Graphical representation of $\text{Latency}_{\text{pp}}(\mathcal{A})(n, \lambda)$ for uniform algorithms.

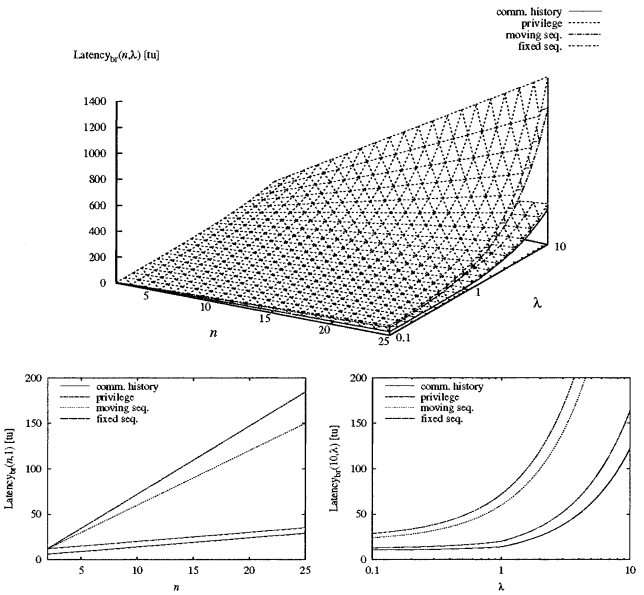


Fig. 9 Graphical representation of $\text{Latency}_{\text{br}}(\mathcal{A})(n, \lambda)$ for uniform algorithms.

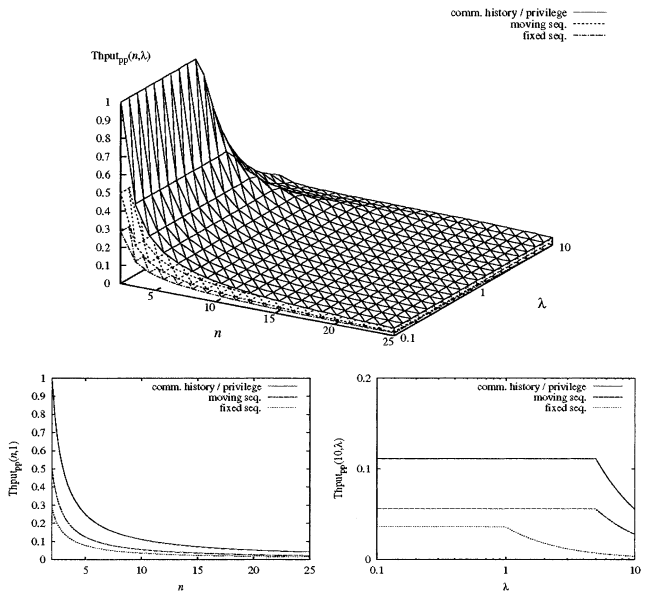


Fig. 8 Graphical representation of $\text{Thput}_{\text{pp}}(\mathcal{A})(n, \lambda)$ for uniform algorithms.

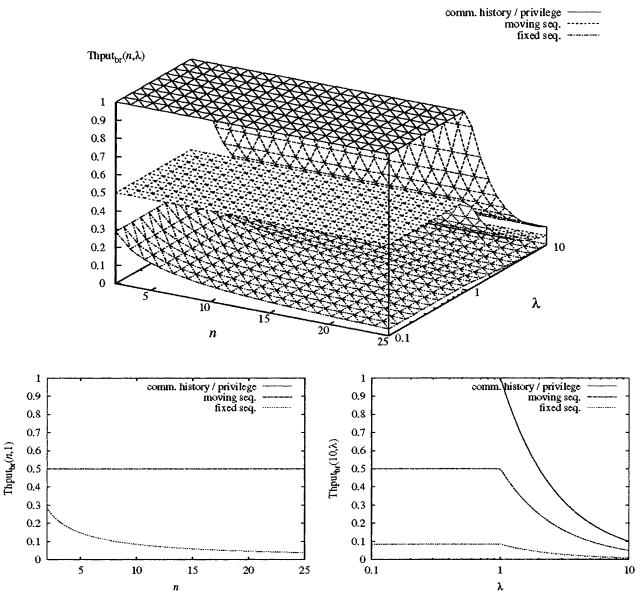


Fig. 10 Graphical representation of $\text{Thput}_{\text{br}}(\mathcal{A})(n, \lambda)$ for uniform algorithms.

fact that, despite the quadratic number of messages, the algorithm generates only little network contention with such a small number of processes.

As a second observation, Fig. 7 (lower right) shows something unexpected: the communication history algorithm has the best latency of all four algorithms, for large values of λ (larger than 4 for $n = 10$). A plausible explanation is that the communication history algorithm has a decentralized communication pattern. This increases the parallelism between processes and thus reduces potential waiting delays. Also, a large value of λ reduces the importance of the network on the overall performance. The network contention is thus the weak point of the communication history algorithm.

The third observation is less obvious, and concerns the privilege-based and the moving sequencer algorithms. The two algorithms have a similar latency. In general, the moving sequencer algorithm seems to slightly outperform the privilege-based algorithm; unlike what time complexity suggests. Nevertheless, Fig. 7 (top; lower right) shows that, for small values of λ, this is reversed and the privilege-based algorithm has a slightly better latency than the moving sequencer algorithm. Figure 7 (lower left) shows that the privilege-based algorithm has a better latency also when the number of processes becomes large ($n > 20$ with $\lambda = 1$).

6.1.2 Thput_{pp}(Uniform Atomic Broadcast)(n, λ)

Figure 8 shows the throughput of the uniform algorithms. The communication history and the privilege-based algorithms can achieve the best throughput among the uniform algorithms. The communication history algorithms does not generate empty messages under a high and evenly distributed load. Consequently, the algorithm effectively generates only one single "send to all" for each application message. Similarly, the privilege-based algorithm generates the same amount of traffic because the high load makes it possible to piggy-back all token messages.

(best) history
= privilege > moving seq. > fixed seq. (worst)

Figure 8 also shows that the two sequencer algorithms have a bad throughput. It is interesting to note that the moving sequencer algorithm always performs slightly better than the fixed sequencer one. This is easily explained by the fact that the moving sequencer uses a token-passing mechanism for the stabilization of messages. It turns out that token messages can easily benefit from the moving sequencer and be piggy-backed on other messages. In contrast, the centralized acknowledgment scheme used by the fixed sequencer makes it more difficult to reduce its overhead in this way.

Figure 8 (lower right) shows that the throughput of all uniform algorithms drops when λ increases over a certain threshold. This threshold depends however on the algorithm: when $n = 10$, the throughput of the fixed sequencer algorithm starts to drop when λ is greater than 1, while the throughput begins to decrease when λ is greater than 5 for the three other algorithms. This shows that the fixed sequencer is more sensitive to a lack of CPU resources than the other algorithms.

Although based on a similar approach, it is interesting to note that the moving sequencer and the fixed sequencer algorithm do not behave in the same way when λ increases. Indeed, as the parameter λ increases, the throughput of the moving sequencer algorithm begins to drop at a later point than for the fixed sequencer algorithm (see Fig. 8; lower right). This is due to the fact that the moving sequencer algorithm distributes the load of sequencing messages evenly among all processes. Conversely, the fixed sequencer algorithm concentrates this load on a single process (the sequencer). It follows that the CPU of the sequencer becomes a bottleneck when the value of λ increases beyond 1.

6.2 Broadcast Networks

6.2.1 $\text{Latency}_{\text{br}}$(Uniform Atomic Broadcast)(n, λ)

Figure 9 shows the latency of the uniform algorithms in a broadcast network. The evaluations obtained with $\text{Latency}_{\text{br}}(\mathcal{A})(n, \lambda)$ are close to those obtained with time complexity. The reason is that the algorithms generate much less traffic than in a point-to-point network, and thus cause only little network contention.

(best) history < fixed seq.
< moving seq. < privilege (worst)

It is interesting to note that, in a broadcast network, the communication history algorithm has a better latency than even the fixed sequencer algorithm. Fixed sequencer are normally considered to have the best latency, but this is mostly because one usually consider the non-uniform algorithm. Here, we see that the cost of uniformity is penalizing for fixed sequencer algorithms. Figure 9 (lower left) shows that the communication history and the fixed sequencer algorithms have a better scalability than the two other algorithms.

A comparison between Fig. 9 and Fig. 7 confirms that the communication history algorithm benefits most from a broadcast network. This is understandable as the algorithm generates a linear number of messages in a broadcast network, whereas its complexity is quadratic in a point-to-point network.

6.2.2 Thput_{br}(Uniform Atomic Broadcast)(n, λ)

Figure 10 shows the throughput of the uniform algorithms in a broadcast network.

(best) history
= privilege > moving seq. > fixed (worst)

The relative performance of the algorithms is the same than in point-to-point networks (compare with Fig. 8). The algorithms however behave differently. For instance, except for the fixed sequencer algorithm, the throughput does not depend on the number of processes (see Fig. 10; lower left). This is a clear evidence that the communication history, the privilege-based, and the moving sequencer algorithms make a better use of the broadcast medium than the fixed sequencer algorithm. The latter is penalized by the positive acknowledgment scheme used for the stabilization of messages.

A comparison between Fig. 10 (lower right) and Fig. 8 (lower right) reveals that, in both cases, the throughput of the algorithms decreases when λ increases beyond a certain threshold. Interestingly, in broadcast networks (Fig. 10), the throughput of *every* algorithm drops when λ increases beyond 1. Among other things, this indicates that the load balancing of the moving sequencer algorithm is of little help in a broadcast network. The weakness of the fixed sequencer algorithm is its stabilization mechanism which leaves few opportunities for piggy-backing messages.

7. Conclusion

We have analyzed four uniform algorithms and four non-uniform Atomic Broadcast algorithms. Each algorithm represents one of the classes of algorithms defined in [8]. The analysis is based on the contention-aware metrics [16], briefly described in Sect. 4. The results obtained through these metrics are significantly more relevant than those obtained with more conventional metrics: time and message complexity. The contention-aware metrics and the conventional metrics also sometimes give opposite predictions.

(1) Uniform algorithms

The analysis of the uniform algorithms gives interesting results. First, the communication history algorithm has the best throughput, but also the best latency in a broadcast network. It also has the best latency in a point-to-point network, if λ is big. On the other hand, this algorithm is poorly scalable in a point-to-point network, with a quadratic degradation of performance as the number of processes increases. We should also point out that the method used to evaluate the throughput is particularly favorable for communication history and privilege-based algorithms.† The privilege-based algorithm has the best throughput (identical to communication history algorithm). However, this algorithm also has the worst latency, except in a point-to-point network and only if the network is more important than the CPU (i.e., if λ is small). The moving sequencer algorithm can be seen as a compromise between throughput and latency. Indeed, the performance of the algorithm is average in all cases. A positive thinking would say that it is never the worst choice, but one could also rightfully claim that it is never the best choice either! Generally speaking, the fixed sequencer algorithm has a good latency which, however, comes at the price of a low throughput. This is already expected from the results of the conventional metrics.

(2) Non-uniform algorithms

With non-uniform algorithms, the best overall results are achieved by the fixed sequencer algorithm. Indeed, this algorithm has the best latency and only comes in second position for the throughput. This comes in clear contrast with the results obtained for the uniform version of the algorithm. This reveals that uniformity comes at a very high price for this particular algorithm. The privilege-based algorithm can achieve a high throughput, but only at the price of latency, for which the algorithm scores third. This algorithm even takes the fourth position in a broadcast network, when the number of processes is large and the importance of the CPU and network resources is roughly equivalent (i.e., λ close to 1). As for its uniform counterpart, the performance of the moving sequencer algorithm is generally average. It nevertheless has the best latency in a broadcast network, identical to the latency of the fixed sequencer algorithm. Finally, the destinations agreement algorithm is the worst both in latency and throughput. It should however be noted that the destinations agreement algorithm can easily be transformed into a uniform algorithm, with no significant performance degradation. Inferring from the results obtained in this chapter, the resulting uniform algorithm would probably have a latency similar to the uniform fixed sequencer algorithm, and a slightly better throughput.

(3) Tradeoffs

Putting the results together yields that the choice of an appropriate algorithm depends on the requirements of the system in which it is to be used. Indeed, if a good throughput is more important than a good latency, then either a communication history or a privilege-based algorithm is probably a good choice. Conversely, if latency is more important than throughput, then a fixed sequencer is better. When both latency and throughput are equally important, a moving sequencer algorithm offers a good compromise.

Of course, there are many more aspects to consider than only performance. The properties of algorithms, or their behavior in the face of failures are indeed often more important than just the question of their raw performance. For instance, if a system requires an open group architecture, then a destinations agreement algorithm may have a good potential in spite of the poor results shown in this study.

†The method used to determine the throughput assumes a high load, evenly distributed among all processes. Hence, the communication history algorithm does not need to generate any empty message, and the privilege-based algorithm can piggy-back all token messages on other messages.

Acknowledgments

We would like to thank the following persons for their insightful comments on early versions of this paper: Adel Cherif, Takuya Katayama, Tohru Kikuno, Jean-Yves Le Boudec, Dahlia Malkhi, Friedemann Mattern, Keith Marzullo, Richard D. Schlichting, Tatsuhiro Tsuchiya, Matthias Wiesmann, as well as the associate editor and the anonymous reviewers.

References

[1] Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, and P. Ciarfella, "The Totem single-ring ordering and membership protocol," ACM Trans. Comput. Syst., vol.13, no.4, pp.311–342, 1995.
[2] K.P. Birman and T.A. Joseph, "Reliable communication in the presence of failures," ACM Trans. Comput. Syst., vol.5, no.1, pp.47–76, 1987.
[3] K.P. Birman, A. Schiper, and P. Stephenson, "Lightweight causal and atomic group multicast," ACM Trans. Comput. Syst., vol.9, no.3, pp.272–314, 1991.

[4] T.D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," J. ACM, vol.43, no.2, pp.225–267, 1996.
[5] J.-M. Chang and N.F. Maxemchuk, "Reliable broadcast protocols," ACM Trans. Comput. Syst., vol.2, no.3, pp.251–273, 1984.
[6] F. Cristian, R. de Beijer, and S. Mishra, "A performance comparison of asynchronous atomic broadcast protocols," Distr. Syst. Eng., vol.1, no.4, pp.177–201, 1994.
[7] F. Cristian, S. Mishra, and G. Alvarez, "High-performance asynchronous atomic broadcast," Distr. Syst. Eng., vol.4, no.2, pp.109–128, 1997.
[8] X. Défago, A. Schiper, and P. Urbán, "Totally ordered broadcast and multicast algorithms: A comprehensive survey," Tech. Rep. DSC/2000/036, Swiss Fed. Inst. Tech., Lausanne, Switzerland, Sept. 2000.
[9] R. Friedman and R. van Renesse, "Packing messages as a tool for boosting the performance of total ordering protocols," Proc. HPDC, pp.233–242, IEEE, Aug. 1997.
[10] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Tech. Rep. CS94-1425, Cornell Univ., USA, May 1994.
[11] M.F. Kaashoek and A.S. Tanenbaum, "Group communication in the Amoeba distributed operating system," Proc. ICDCS, pp.222–230, IEEE, May 1991.
[12] B. Kemme, F. Pedone, G. Alonso, A. Schiper, and M. Wiesmann, "Using optimistic atomic broadcast in transaction processing systems," IEEE Trans. Knowl. Data Eng., vol.15, no.4, pp.1018–1032, 2003.
[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol.21, no.7, pp.558–565, 1978.
[14] L.L. Peterson, N.C. Buchholz, and R.D. Schlichting, "Preserving and using context information in interprocess communication," ACM Trans. Comput. Syst., vol.7, no.3, pp.217–246, 1989.
[15] F.B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," ACM Comput. Surv., vol.22, no.4, pp.299–319, 1990.
[16] P. Urbán, X. Défago, and A. Schiper, "Contention-aware metrics for distributed algorithms," Proc. ICCCN, pp.582–589, IEEE, Oct. 2000.

Appendix: Metric Formulas

Non-Uniform Fixed Sequencer

$$\begin{aligned}
\mathrm{Latency}_{\mathrm{pp}}(\textit{fixed seq.})(n,\lambda) &= 2(2\lambda+1)+(n-2)\max(1,\lambda)\\
\mathrm{Latency}_{\mathrm{br}}(\textit{fixed seq.})(n,\lambda) &= 4\lambda+2\\
\mathrm{Thput}_{\mathrm{pp}}(\textit{fixed seq.})(n,\lambda) &= \frac{n}{(n^2-1)\max(1,\lambda)}\\
\mathrm{Thput}_{\mathrm{br}}(\textit{fixed seq.})(n,\lambda) &= \frac{n}{(2n-1)\max(1,\lambda)}
\end{aligned}$$

Non-Uniform Moving Sequencer

$$\begin{aligned}
&\mathrm{Latency}_{\mathrm{pp}}(\textit{moving seq.})(n,\lambda)\\
&= 2\lambda+2+\begin{cases}
\left\{\begin{array}{ll}\lambda+1 & \text{if } n=2\\ 2n-4 & \text{otherwise}\end{array}\right\} & \text{if } \lambda<\frac{1}{2}\\
\begin{cases}2\lambda+(n-2)\max(1,\lambda) & \text{if } n<4\\ 2\max(1,\lambda) & \\ \quad+\left\{\begin{array}{ll}2n-6 & \text{if } n>\frac{6-2\lambda}{2-\lambda}\\ (n-2)\lambda & \text{otherw.}\end{array}\right\} & \text{otherw.}\end{cases} & \text{if } \frac{1}{2}\le\lambda<2\\
n\lambda & \text{otherwise}
\end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Latency}_{\mathrm{br}}(\textit{moving seq.})(n,\lambda) &= 4\lambda+2\\
\mathrm{Thput}_{\mathrm{pp}}(\textit{moving seq.})(n,\lambda) &= \frac{1}{\max\left(\frac{4n^2-4n-1}{n^2}\lambda,\,2n-2\right)}\\
\mathrm{Thput}_{\mathrm{br}}(\textit{moving seq.})(n,\lambda) &= \frac{1}{2\max(1,\lambda)}
\end{aligned}$$

Non-Uniform Privilege-Based

$$\begin{aligned}
&\mathrm{Latency}_{\mathrm{pp}}(\textit{privilege-based})(n,\lambda)\\
&= \left(\tfrac{n}{2}+1\right)\cdot(2\lambda+1)+(n-2)\cdot\max(1,\lambda)\\
&+\begin{cases}
\max(0,n-3) & \text{if } \lambda\le\frac{1}{2}\\
\max\left(0,\left\lfloor\frac{n-3}{2}\right\rfloor\right) & \text{if } \frac{1}{2}<\lambda\le 1\\
\left\{\begin{array}{ll}\max\left(0,\,\begin{array}{l}2-\lambda\\ +((n-5)\bmod 3)\,(1-\lambda)\end{array}\right) & \text{if } n>4\\ \quad+\max\left(0,\left\lfloor\frac{n-5}{3}\right\rfloor(4-3\lambda)\right) & \\ 0 & \text{otherw.}\end{array}\right\} & \text{if } 1<\lambda\le 2\\
\left\{\begin{array}{ll}1+\lceil\lambda\rceil-\lambda & \text{if } n>2 \text{ and}\\ & (n-2)\bmod(2\lfloor\lambda\rfloor+1)=0\\ 0 & \text{otherwise}\end{array}\right\} & \text{otherwise}
\end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Latency}_{\mathrm{br}}(\textit{privilege-based})(n,\lambda) &= \left(\frac{n}{2}+1\right)\cdot(2\lambda+1)\\
\mathrm{Thput}_{\mathrm{pp}}(\textit{privilege-based})(n,\lambda) &= \frac{1}{(n-1)\cdot\max\left(1,\frac{2\lambda}{n}\right)}\\
\mathrm{Thput}_{\mathrm{br}}(\textit{privilege-based})(n,\lambda) &= \frac{1}{\max(1,\lambda)}
\end{aligned}$$

Non-Uniform Destinations Agreement

$$\begin{aligned}
&\mathrm{Latency}_{\mathrm{pp}}(\textit{dest. agreement})(n,\lambda)\\
&= \begin{cases}
3(n-1)+4\lambda+2\lambda\,((n-1)\bmod 2) & \text{if } \lambda<\frac{1}{2}\\
3(n-1)+4\lambda+\left\{\begin{array}{ll}2\lambda & \text{if } n\bmod 3=2\\ 2\lambda-1 & \text{if } n\bmod 3=0\\ 0 & \text{otherwise}\end{array}\right\} & \text{if } \frac{1}{2}\le\lambda<1\\
(3n-2)\lambda+1 & \\
\quad+\left\{\begin{array}{ll}2+(4-n)\lambda & \text{if } n<5\\ 4-2\lambda & \text{if } n=5\\ \max\left(0,\,\begin{array}{l}\left\{\begin{array}{ll}\lambda+1 & \text{if } n\bmod 4=0\\ 2 & \text{if } n\bmod 4=1\\ 3 & \text{otherwise}\end{array}\right\}\\ -(n-4)(2\lambda-2)\end{array}\right) & \text{otherw.}\end{array}\right\} & \text{if } 1\le\lambda<\frac{3}{2}\\
(3n-2)\lambda+1+\left\{\begin{array}{ll}2+(4-n)\lambda & \text{if } n<5\\ \max(0,4-2\lambda) & \text{if } n=5\\ 0 & \text{otherwise}\end{array}\right\} & \text{otherwise}
\end{cases}
\end{aligned}$$

$$\begin{aligned}
\mathrm{Latency}_{\mathrm{br}}(\textit{dest. agreement})(n,\lambda) &= 6\lambda+3+(n-2)\max(1,\lambda)\\
\mathrm{Thput}_{\mathrm{pp}}(\textit{dest. agreement})(n,\lambda) &= \frac{1}{(3n-3)\max\left(1,\frac{2\lambda}{n}\right)}\\
\mathrm{Thput}_{\mathrm{br}}(\textit{dest. agreement})(n,\lambda) &= \frac{1}{\max\left(n+1,\frac{4n-2}{n}\lambda\right)}
\end{aligned}$$

Uniform Fixed Sequencer

$$\begin{aligned}
&\mathrm{Latency}_{\mathrm{pp}}(\textit{unif. fixed seq.})(n,\lambda)\\
&\quad= 2\lambda+1+\mathrm{Latency}_{\mathrm{pp}}(\textit{dest. agreement})(n,\lambda)
\end{aligned}$$

$$\begin{aligned}
\mathrm{Latency}_{\mathrm{br}}(\textit{unif. fixed seq.})(n,\lambda) &= 8\lambda+4+(n-2)\max(1,\lambda)\\
\mathrm{Thput}_{\mathrm{pp}}(\textit{unif. fixed seq.})(n,\lambda) &= \frac{n}{(3n^2-2n-1)\max(1,\lambda)}\\
\mathrm{Thput}_{\mathrm{br}}(\textit{unif. fixed seq.})(n,\lambda) &= \frac{n}{(n^2+2n-1)\max(1,\lambda)}
\end{aligned}$$

Uniform Moving Sequencer

$$\begin{aligned}
&\text{Latency}_{\text{pp}}(\textit{unif. moving seq.})(n,\lambda)\\
&= 2(n-1)(2\lambda+1)\\
&+\begin{cases}
\begin{cases}
2(n-\lambda-1) & \text{if } n>4\\
\begin{cases}
\max(2n-4,4\lambda)+2 & \text{if } \lambda \geq \frac{1}{2}\\
\begin{cases}
6-2\lambda & \text{if } n=4\\
4 & \text{if } n=3\\
2+4\lambda & \text{if } n=2
\end{cases} & \text{otherw.}
\end{cases} & \text{otherw.}
\end{cases} & \text{if } \lambda<1\\
\begin{cases}
2\lambda+\max\left((n-2)\lambda, 2\lambda+2\right)\\
\quad+\begin{cases}
2n-6\\ \quad-(n-2)\lambda & \text{if } n \geq \frac{6-2\lambda}{2-\lambda}\\
2-\lambda & \text{if } n < \frac{5-2\lambda}{2-\lambda}\\
2n-5\\ \quad-(n-2)\lambda & \text{otherwise}
\end{cases} & \text{if } n>5\\
2\lambda+1+\max(2\lambda+1, 2+\lceil\lambda\rceil) & \text{if } n=5\\
4\lambda+2 & \text{otherw.}
\end{cases} & \text{if } 1\leq\lambda<2\\
2\lambda+\max\left((n-2)\lambda, 2\lambda+2\right) & \text{otherwise}
\end{cases}
\end{aligned}$$

$$\begin{aligned}
\text{Latency}_{\text{br}}(\textit{unif. moving seq.})(n,\lambda) &= 4n\lambda+2n\\
\text{Thput}_{\text{pp}}(\textit{unif. moving seq.})(n,\lambda) &= \frac{1}{\max\left(\frac{4n^2-4n-1}{n^2}\lambda, 2n-2\right)}\\
\text{Thput}_{\text{br}}(\textit{unif. moving seq.})(n,\lambda) &= \frac{1}{2\max(1,\lambda)}
\end{aligned}$$

Uniform Privilege-Based

$$\begin{aligned}
&\text{Latency}_{\text{pp}}(\textit{unif. privilege-based})(n,\lambda)\\
&= \tfrac{5n}{2}(2\lambda+1)+\begin{cases}
(n-2)(1-2\lambda) & \text{if } \lambda \leq \frac{1}{2}\\
\left\lfloor\frac{n-2}{2}\right\rfloor(2-2\lambda) & \text{if } \frac{1}{2}<\lambda\leq 1\\
(3-2\lambda)\max\left(0, \left\lfloor\frac{n-4}{3}\right\rfloor\right) & \text{if } 1<\lambda\leq\frac{3}{2}\\
(\lambda-\lfloor\lambda\rfloor)\max\left(0, \left\lfloor\frac{n-4}{3}\right\rfloor\right) & \text{if } \frac{3}{2}<\lambda\leq 2\\
(\lambda-\lfloor\lambda\rfloor)\max\left(0, \left\lfloor\frac{n-3}{5}\right\rfloor\right) & \text{otherwise}
\end{cases}
\end{aligned}$$

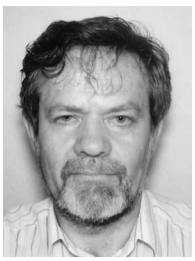
$$\begin{aligned}
\text{Latency}_{\text{br}}(\textit{unif. privilege-based})(n,\lambda) &= \left(\frac{5n}{2}-1\right)\cdot(2\lambda+1)\\
\text{Thput}_{\text{pp}}(\textit{unif. privilege-based})(n,\lambda) &= \frac{1}{(n-1)\max\left(1,\frac{2\lambda}{n}\right)}\\
\text{Thput}_{\text{br}}(\textit{unif. privilege-based})(n,\lambda) &= \frac{1}{\max(1,\lambda)}
\end{aligned}$$

Uniform Communication History

$$\begin{aligned}
&\text{Latency}_{\text{pp}}(\textit{unif. comm. history})(n,\lambda)\\
&=\begin{cases}
4\lambda+2 & \text{if } n=2\\
6\lambda+2+\max(0,2-\lambda)\\ \quad+\max(0,1-\lambda)+\max(0,1-2\lambda) & \text{if } n=3\\
3(n-1)\lambda+1 & \text{if } n\leq\lambda+2\\
\frac{n^2-3n}{2}+2n\lambda+\lfloor\lambda\rfloor\lambda-\frac{\lfloor\lambda+3\rfloor\cdot\lfloor\lambda\rfloor}{2}\\ \quad+\begin{cases}1 & \text{if } \lfloor\lambda\rfloor=n-3\\ 0 & \text{otherwise}\end{cases} & \text{if } n\leq 2\lfloor\lambda\rfloor+3 \text{ and } \lambda\geq 3\\
\frac{n^2-n}{2}+2n\lambda+\lfloor\lambda\rfloor\lambda-\frac{\lfloor\lambda+3\rfloor\cdot\lfloor\lambda\rfloor}{2}-\lfloor 2\lambda\rfloor-3\\ \quad+\begin{cases}
\lfloor 2\lambda-1\rfloor & \text{if } n=5\\
2 & \text{if } n=\lfloor 2\lambda+1\rfloor\\
1 & \text{if } n=7 \text{ and } \lambda=\frac{11}{4}\\
0 & \text{otherwise}
\end{cases} & \text{if } n\leq 4\lambda-4\\
n^2-n\\ \quad+\begin{cases}
2\lambda & \text{if } \lambda<1\\
\begin{cases}5\lambda-3 & \text{if } n=4\\ 4\lambda-2 & \text{if } n>4\end{cases} & \text{if } 1\leq\lambda<2\\
5\lambda-4 & \text{if } 2\leq\lambda<3\\
2\lfloor\lambda\rfloor\lambda-\lfloor\lambda\rfloor-\lfloor\lambda\rfloor^2+5 & \text{otherwise}
\end{cases} & \text{otherwise}
\end{cases}
\end{aligned}$$

$$\begin{aligned}
\text{Latency}_{\text{br}}(\textit{unif. comm. history})(n,\lambda) &= 4\lambda+2+(n-2)\max(1,\lambda)\\
\text{Thput}_{\text{pp}}(\textit{unif. comm. history})(n,\lambda) &= \frac{1}{(n-1)\cdot\max\left(1,\frac{2\lambda}{n}\right)}\\
\text{Thput}_{\text{br}}(\textit{unif. comm. history})(n,\lambda) &= \frac{1}{\max(1,\lambda)}
\end{aligned}$$

Xavier Défago is (research) associate professor at the Japan Advanced Institute of Science and Technology (JAIST) since 2003, and a researcher for the PRESTO program "Information and Systems" of the Japan Science and Technology Corporation (JST) since 2002. He obtained his Ph.D. in Computer Science in 2000 from the Swiss Federal Institute of Technology in Lausanne (EPFL; Switzerland), and joined JAIST as a research associate shortly thereafter. From 1995 to 1996, he also worked at the NEC C&C Research Labs in Kawasaki (Japan). His research interests include distributed algorithms, fault tolerance, large-scale distributed systems, group communication, and cooperative autonomous mobile systems.



André Schiper has been professor of Computer Science at the EPFL (Swiss Federal Institute of Technology in Lausanne) since 1985, leading the Distributed Systems Laboratory. During the academic year 1992-93 he was on sabbatical leave at the University of Cornell, Ithaca (NY). His research interests are in the areas of fault-tolerant distributed systems, middleware, group communication, and recently mobile ad-hoc networks (Swiss MICS project). He took part in the following European projects: ESPRIT BROADCAST (1992-1995, 1996-1999), ESPRIT R&D OpenDREAMS I and II (1996, 1997-1999), IST REMUNE (2001-2003). He is member of the IST Network of Excellence in Distributed and Dependable Computing Systems (CABERNET). From 2000 to 2002, he was the chair of the steering committee of the International Symposium on Distributed Computing (DISC).

Péter Urbán is a research and teaching assistant at the Distributed Systems Laboratory at the Swiss Federal Institute of Technology in Lausanne (EPFL). He obtained his Ph.D. in Computer Science in 2003 from the Swiss Federal Institute of Technology in Lausanne (EPFL; Switzerland), and the M.Sc. degree in Computer Science in 1997 from the Budapest University of Technology and Economics (Hungary). From 1997 to 1998, he worked at the CERN European Laboratory for Particle Physics in Geneva, Switzerland. His research interests include distributed systems, middleware, fault tolerance and performance evaluation.