An Integrated Group Communication Infrastructure for Hybrid Real-time Distributed Systems

Article

2 authors, including:

Raimundo J de A Macêdo
Universidade Federal da Bahia
78 PUBLICATIONS **650** CITATIONS

SEE PROFILE

An Integrated Group Communication Infrastructure for Hybrid Real-time Distributed Systems

Raimundo José de Araújo Macêdo

Distributed Systems Laboratory (LaSiD), Computer Science Department, Federal University of Bahia – Campus de Ondina. Salvador, Bahia, Brasil

macedo@ufba.br

Abstract. Group Communication is a powerful abstraction that is being widely used to manage consistency problems in a variety distributed system models, ranging from synchronous, to time-free asynchronous model. Though similar in principles, distinct implementation mechanisms have been employed in the design of group communication for distinct system models. However, the hybrid nature of many modern (real-time) distributed systems, with distinct QoS guarantees, has put forward the need for integrated models, where both hard and soft real-time application services must be provided within the very same system. Moreover, in such scenarios, adaptation with degraded service is a common requirement. This paper tackles this new challenge by introducing a mechanism called the Timed Causal Blocks, which is an extension of the so-called Causal Blocks model used for asynchronous systems. Because they combine physical and logical time in the same infrastructure, Timed Causal Blocks represents an integrated framework capable of handling group communication for both synchronous hard real-time and asynchronous soft real-time distributed systems. This paper describes the information structure of Timed Causal Blocks and sketches the related algorithms intended to total order message delivery and dynamic membership reconfiguration. Moreover, it is shown that the ordering and reliability information provided by the Causal Blocks can be used to anticipate message delivery, and timely delivery can be achieved without relying on clock synchronization.

Key works: group communication, fault tolerance, asynchronous and synchronous systems.

1. Introduction

Group communication is a powerful abstraction that can be used whenever groups of distributed processes cooperate for the execution of a given task [1,3-6,10]. In group communication, processes communicate in a group basis where a message is sent to a group of processes. With a group is usually associated a name to which application processes will refer, making transparent the location of the distributed processes forming the group. Due to the uncertainties inherent to distributed systems (emerging from communication or process failures), group communication protocols have to face situations where, for instance, a sender process fails when a multicast is underway or where messages arrive in an inconsistent order at different destination processes. Moreover, a consistent view of the set of functioning processes is fundamental for some applications. For instance, when server replication is employed to attain high dependability of services, surviving server replicas must take over the responsibilities of failed ones. Therefore, a central problem that has to be solved in such a context is to maintain consistent the sever replica states and the views that distinct replicas have about the set of functioning ones. Group

Communication is being widely used to manage such problems in a variety distributed system models, ranging from synchronous, to time-free asynchronous model.

Related work on distributed systems models for faul tolerance and real-time

Distributed systems can be broadly classified into two categories: synchronous and time-free asynchronous. In synchronous systems, message transmission and process execution delays are bounded, and, in most cases, these bounds are assumed to be known in advance. Synchronous model is natural for hard real-time distributed systems, since it guarantees bounded reaction time for events. This model assumption simplifies the treatment of failures since a process failing to send a message (or processing it) within the bounded delay can be considered to have failed. As a consequence, several problems related to fault-tolerant computing, such as membership, consensus, and atomic broadcast have been solved in such a model. The price that has to be paid is the real-time scheduling and hardware redundancy techniques that need to be used to guarantee that such a bounded time assumption is achieved with high probability [11-15].

In an asynchronous system, on the other hand, there is no known bound for message transmission or processing times. This makes the system more portable and less sensitive to operational conditions (for example, long unpredictable transmission times will not affect safety properties of the system). However, it is well known that some fundamental fault-tolerant problems have no deterministic solution in such a model (e.g., the consensus problem [9]). The time-free asynchronous model is a rather pessimistic model, however. In practice, most systems (specially those built from off-the-shelf components) are neither fully synchronous, nor fully asynchronous. Most of the time they behave synchronously, but can have “unstable" periods during which they behave in an anarchic way. That is way many researches have successfully identified distinct stability conditions necessary to solve fundamental fault-tolerant problems [17-18]. Other researches have considered systems composed by a synchronous and an asynchronous part at the same time. So, we can regard such systems as being hybrid in the space dimension. This is the case of the TCB model, which relies on a synchronous wormhole to implement fault tolerant services [16]. The concept of real-time dependable channels (RTD) that has been introduced in [20] to handle QoS adaptation at channel creation time is another example. In their paper, the authors show how an application can customize a RTD channel according to specific QoS requirements, such as bounded delivery time, reliable delivery, message ordering, and jitter (variance in message transmission time). Such QoS requirements are stated in terms of the probabilities and are enforced by a composite protocol based on the real-time version of the Cactus system. Resource reservation and admission control have been used in “QoS architectures" in order to allow processes to dynamically negotiate the quality of service of their communication channels, leading to settings with hybrid characteristics that can change over time [7]. A similar approach has been taken in a real-time group communication protocol called RTCAST, where a resource reservation scheme is used to ensure predictable access to system resources in non-real-time operating systems, increasing the timeliness predictability of such environments [19].

Contributions of this paper

A challenge not adequately addressed so far is how to combine the advantages of asynchronous and synchronous models, exploring the hybrid nature of such existing distributed systems, to construct hybrid group communication facilities, with adaptive characteristics. As has been presented in [15], though similar in principles, distinct strategies have been adopted to group communication, depending on the system environment (synchronous or asynchronous) and application requirements (timely message delivery, total and causal message ordering, etc). Therefore, the design of integrated group communication schemes, which can work on hybrid

models, with adaptiveness characteristics, remains a challenge in the area. The present paper tackles this new challenge by introducing a mechanism called the Timed Causal Blocks (TimedCB), which is an extension of the so-called Causal Blocks model used for asynchronous systems. The Causal Blocks is a framework for developing group communication protocols and related services with a number of ordering and reliability properties, such as ordered message delivery in overlapping groups, flow control, etc. [3-6]. Because they combine physical clock time and logical time in the same infrastructure, TimedCB represents an integrated framework capable of handling group communication for both synchronous hard real-time and asynchronous soft real-time distributed systems. This is especially relevant to achieve dynamic adaptation (one could switch to the asynchronous version when timely conditions can no longer be met) and fast message delivery (for instance, there is no need to wait for a timing condition when some logical time property is already satisfied within a time window – for example, for timely total ordered message delivery). This paper describes the information structure of TimedCB and sketches the related algorithms intended to total order message delivery. The remaining of this paper is structured as follows. Section 2 presents the TimedCB framework and related total order message delivery conditions, and section 3 draws some conclusions.

2.0 The Timed Causal Blocks (TimedCB) Framework

2.1 The Underlying System Model and Assumptions

It is assumed that the underlying system is capable of providing distinct QoS guarantees for both message diffusion and process scheduling times. Moreover, it is assumed that the underlying system behavior can change over time, such that processes and channels may alternate their QoS. This may happen due to QoS renegotiations and/or failures.

A system consists of a finite set Π of $n > 1$ processes, namely, $\Pi = \{p_1, p_2 \ldots, p_n\}$. Processes communicate and synchronize by sending and receiving multicast messages through channels, and every pair of processes (p_i, p_j) is connected by a reliable bidirectional channel: they do not create, alter, or lose messages. In particular, if p_i sends a message to p_j, then if p_i is correct, eventually p_j receives that message unless it fails. Transmitted messages are received in FIFO order. Processes form a unique group g, whose initial configuration is Π. That is, $g = \{p_1, p_2 \ldots, p_n\}$. Due to space limitations, we do not consider multiple groups in this paper. Processes have access to local hardware clocks with drift rate ρ.

A process executes steps (a step is the reception of a message, the sending of a message, each with the corresponding local state change, or a simple local state change). The system observed for processes and channels lead to the identification of two sub-sets of Π: S and A, the set of processes considered synchronous and asynchronous, respectively. That is, if process $p \in S$, there is a time bound ϕ for a computation step executed by p, and p is necessarily connected to (at least) another synchronous process p_j, by a bidirectional channel with known maximum and minimum bounds for message transmission times, denoted Δ_{max} and $\Delta_{min,}$ respectively[1]. If these conditions are not satisfied, processes are considered asynchronous, that is, they belong to A. Sets A and S form a partition of Π (i.e., $A \cup S = \Pi$ and $A \cap S = \varnothing$). In the formulas, we assume that $\Delta >> \phi$, so ϕ can be neglected. That is, the time to transmit messages is much larger than the time to process them, so we can ignore ϕ for the sake of message delivery time calculations. It is also assumed that the underlying system is equipped with a monitoring mechanism that provides processes with the information of the current QoS guarantee ensured for a given channel or

[1] Observe that a process can only execute synchronous distributed computations if it is connected by timely channels with other cooperating processes.

process (the change of status can be informed within some time lag and two distinct processes are not required to have the same view of the QoS related to a third process at a given time). Such monitoring mechanisms are in charge of updating the sets S and A, and applications process can only read the contents of S and A. The sets S and A are similar to the *live* and *uncertain* sets presented in [8], respectively. More precisely, *live* $\subseteq$ S and A = *uncertain*. Accordingly, the QoS provider mechanism presented in [8] is similar to the monitoring mechanism assumed in this paper.
We do not consider failures in describing the basic TimedCB mechanism (due to space limitations). Later, in this text, we comment on the treatment of process crash failures.

2.2 An Overview of the Causal Blocks Model

Each process p_i maintains a logical clock called the Block Counter and denoted as BC_i. BC_i is an integer variable and its value increases monotonically. When g is created, the BC_i of every p_i is initialized to any finite integer, and without loss of generality, we will suppose that they are all initialized to zero. Transmitted messages are timestamped with Block Counters, and, as it is the case in Lamport's Logical Clock, timestamping using Block Counters will respect causality. That is, if m $\rightarrow$ m', then the timestamp associated with m' will be larger than the timestamp associated with m. However, the reverse will not always hold. If two messages have different timestamp values, they may be concurrent. Unlike Lamport's Logical Clock that is advanced on send/receive events, Block Counters advances on send/delivery events, since a transmitted message can only be causally dependent on previously delivered ones.

Just before a process p_i multicasts a message m, it advances BC_i by one. The contents of the incremented BC_i is assigned to m as its block-number in the message field m.b. BC_i may also be advanced by p_i on $delivery_i(m)$ if the current value of BC_i is less than m.b. Thus, the two events under which BC_i may be advanced are:

CA1 (Counter advances during $send_i(m)$): Before p_i multicasts m, it increments BC_i by one, and assigns the incremented value to m.b; and,

CA2 (Counter advances during $delivery_i(m)$): Before pi delivers m, it sets $BC_i = \max\{BC_i, m.b\}$.

Based on *CA1* and *CA2* we can state the three following properties possessed by block-numbers of multicast messages. For notational simplicity, we denote $send_{pi}$ as simply $send_i$.

pr1 : $send_i(m) \rightarrow send_i(m') \Rightarrow m.b < m'.b$.

pr2 : for any *m*, P_j $m.g : deliver_j(m) \rightarrow send_j(m') \Rightarrow m.b < m'.b$; and,

pr3 : for all $m', m'' : m'.b = m''.b \Rightarrow m'$ and m'' are concurrent.

The properties *pr1* and *pr2* follow directly from *CA1* and *CA2*, respectively. Together they imply that for any distinct m, m': send(m) $\rightarrow$ send(m') $\Rightarrow$ $m.b < m'.b$. The property *pr3* states that distinct messages multicast with the same block-number are necessarily concurrent and these messages must have been multicast by distinct processes, as *CA1* forbids two send events to occur in a given process with the same value of BC.

2.2.1 Causal Blocks and the Block Matrix

Using *pr3*, p_i constructs Causal Blocks to represent concurrent messages it sent/received with the same block-number. Construction of Causal Blocks leads to the notion of Block Matrix which can be viewed as a convenient way of representing sent and received messages with different block-numbers. A Causal Block is a vector of size equal to n = |g|. Whenever a process p_i receives or sends a multicast message with a new block-number, say B, it creates an empty vector of length n; for any message multicast with block-number B, it sets the i^{th} entry of the vector to '+'; and, for any multicast message received with block-number B from another process

p_j, $j \neq i$, it sets the j^{th} entry of the vector to '+'. Causal Blocks, maintained by a process in this way, will have the following property:

PR1 : in a given Causal Block, only concurrent messages are represented.

When group communication is active, i.e. member processes continually send multicast messages, the number of Causal Blocks constructed will grow. Causal Blocks maintained by a process are arranged in the increasing order of the message block-number they represent, giving rise to a matrix which is called the *Block Matrix* and denoted as BM. Thus, BM[B] will represent the Causal Block for message block-number B. Referring to BM of any member process in G, we can state another property of Causal Blocks,

PR2: if m and m', represented in BM[B] and BM[B'], respectively, are causally related such that $m \rightarrow m'$, then $B < B'$.

	p_1	p_2	p_3	p_4	p_5	p_6
1	+			+		
2		+			+	+
3	+		+	+		
4	+				+	
5		+				

Figure 1 - The Block Matrix of a 6-member Group Process

Figure 1 shows the Block Matrix of a 6-member process group. It represents all messages sent/received by the process which owns this particular matrix. Since messages arrive at destinations with presumably distinct delays, the BM matrices of different processes will not necessarily be the same in a given moment of physical time. The BM matrix showed in figure 2, indicates, for example, that the block-numbers of the last messages received from processes p_1 and p_2, are 4 and 5, respectively.

2.3 Timely and Untimely Block Completion

By *PR1*, all messages which belong to a given Causal Block, say BM[B], are concurrent. From *PR2*, any causal order message delivery based solely on block numbers will require that a received message represented in BM[B], $B > 1$, be delivered only after all multicast messages which can be represented in BM[B'], for all $B' < B$, have been delivered.

To enable a member process to accurately determine that a given block completely represents all messages which can be represented in it, we use the notion of *block completion*.

Logical Time Block Completion. A Causal Block BM[B], $B \geq 1$, will be said to be *complete*, if and only if for all j, $1 \leq j \leq n$, the j^{th} entry of BM[B] either (i) has '+' or (ii) is blank and there exists B', B'>B such that the j^{th} entry of BM[B'] has '+'. By (i), it is ensured that if p_j has sent a message with block-number B, it is received. By *CA1* and *CA2*, p_j will not produce two distinct multicast messages with the same block-number. Since messages from a given sender process are received in FIFO order, (ii) guarantees that p_j has not multicast any message with block-number B. In the example shown in figure 2, block 2 is complete because processes p_2, p_5, and p_6 have sent a message with block-number 2, and processes p_1, p_3, and p_4 have sent a message with block-number 3.

From *PR2* it is obvious that the existence of BM[B], $B > 1$, also implies the existence of some Causal Block BM[B'], for some $B' < B$ ($B' = B - 1$ if Block Counters are incremented by 1). Since messages are assumed to be transmitted in FIFO order, Causal Blocks in the BM maintained by a process will complete in the sequentially increasing order of block-numbers they represent.

A given causal block is guaranteed to complete only if processes in g remain lively by sending messages so that block counter increases with time: a necessary condition to deliver ordered messages (as will be seen below). To accomplish that, the Causal Blocks model provides each process with a simple mechanism, called the time-silence, which enables a process to remain lively during those periods when the process is required to be lively but is not generating computational messages.

Time-silence Mechanism

The time-silence mechanism of p_i, $timesilence_i$, works as follows. A process p_i maintains in the integer variables $SENT_i$ and $RECD_i$, the largest block number of the messages p_i has sent and received at any given time, respectively. Whenever p_i receives a multicast message with block number β, it checks whether $SENT_i \geq \beta$. If $SENT_i < \beta$, then a local-time-silence timeout, denoted as $ts(\beta)$, for some predetermined time period is set. This timeout period indicates the duration within which p_{pi} is expected to multicast a message with a block-number β or larger - thus, making its contribution towards the delivery of all m, $m.b \leq \beta$, by all $p_j \in g$ (including itself). When $ts(\beta)$ expires, p_i multicasts a null message m with the block-number $m.b = RECD_i$ if $SENT_i < \beta$ is still true; if $SENT_i \geq \beta$, the expiry of $timeout(\beta)$ is ignored. Note that by multicasting a null message with block-number $m.b = RECD_i$, Pi contributes to the delivery of all m, $m.b \leq RECD_i$. Stated below is the third possibility CA3 by which the BC_i is advanced due ti $timesilence_i$.

CA3(Counter Advance due to timesilencei): Before p_i multicasts a null message, it sets $m.b = RECD_i$ and $BC_i = m.b$.

Finally, we observe that the time-silence mechanism of a given p_i ensures that $SENT_i = RECD_i$ becomes true in finite time if $RECD_i$ ever becomes larger than $SENT_i$. The above block completion definition has been used to implement asynchronous protocols in the Causal Blocks framework, as it does not take into consideration message transmission times. We now introduce the notion of timely block completion, meaning the upper bound time by which a created causal block will complete.

Timely Block Completion. A causal block BM[m.b] is created for a process p_i either when p_i sends or when p_i receives m and BM[m.b] has not been created yet. First let us consider that BM[m.b] is created for p_i at local time t_i when p_i sends m. In this scenario, p_i will have to wait until time $t_i + ts(m.b) + 2\Delta_{max}$ to receive a message (non-null or null) from every p_j, $i \neq j$. The first Δ_{max} is due to the time necessary for *m* to reach p_j, and ts(m.b) the time bound for p_j to send a message for BM[m.b]. Finally, the other Δ_{max} is the time bound to p_i to receive the message generated by p_j for BM[m.b].

Now consider that BM[m.b] is created for p_i at local time t_i when p_i receives m from a process p_j. Soon after BM[m.b] is created at p_i, ts(m.b) is set up to expire at $t_i + ts(m.b)$ - as a time bound for p_i to send a message for BM[m.b]. As every other p_j, $i \neq j$, also receives m (reliable channel assumption) and sets ts(m.b), p_i will have to wait until $t_i + (\Delta_{max} - \Delta_{min}) + ts(m.b) + \Delta_{max}$. The difference $(\Delta_{max} - \Delta_{min})$ is to account for the extreme case where a process p_j may receive *m* in Δ_{max} and p_i receives *m* in Δ_{min}. ts(m.b) is the time necessary for p_j to send a message in BM[m.b] and Δ_{max} the time necessary for the message to travel from p_j to p_i. Hence, the time bounds for a BM[m.b] to complete at a process p_i, as measured by its local clock is:

- TC1: $((t_i + ts(m.b) + 2\Delta_{max})(1 + \rho))$, if m was sent by p_i
- TC2: $((t_i + 2\Delta_{max} - \Delta_{min} + ts(m.b))(1 + \rho))$, if m was received by p_i

And a Causal Block BM[m.b] is complete at p_i as soon as one of the conditions below (or both) occurs:

i) BM[m.b] is timely complete
ii) BM[m.b] is logical time complete

Total Order Message Delivery. Total order delivery ensures that all group members deliver the same set of messages and in the same order. The message delivery conditions for a process p_i are stated below:

- safe1: a received m, is deliverable if BM[m.b] is complete;
- safe2: deliverable messages are delivered in the non-decreasing order of their block numbers; a fixed pre-determined delivery order is imposed on deliverable messages of equal block number.

The two safety conditions ensure that the received messages are delivered in total order when they become deliverable, as long as processes do not fail. Suppose now that a process p_k fails by stop functioning (crash) and, as a consequence, the block completion timeouts (TC1 or TC2) expire at p_i for a BM[m.b]. In order to proceed with message delivery, a new membership for g must be established that excludes p_k (or any other processes that did not contribute for the block completion). The generic consensus protocol presented in [8] can be used to attain this new membership.

The Timed Causal Blocks for the Hybrid Model. So far, we have considered that all processes belong to S, which characterizes a synchronous system with the Δ_{max} - Δ_{min} assumption. If that is not is the case, it does not make sense to apply the conditions TC1 and TC1. However, the conditions TC1 and TC2 can be slightly modified so that the completion test can be selected according to the synchrony of the process. Thus, the time bounds for a BM[m.b] to complete in a hybrid system (i.e., $S \neq \emptyset$ and $A \neq \emptyset$), at a process p_i, as measured by its local clock, is:

If $p_i \in A$, then check if BM[m.b] is logical time complete
If $p_i \in S$
for every $p_j \in S$, $j \neq i$, wait for the following times:

- TC1: $((t_i + ts(m.b) + 2\Delta_{max})(1 + \rho))$, if m was sent by p_i
- TC2: $((t_i + 2\Delta_{max} - \Delta_{min} + ts(m.b)(1 + \rho))$, if is m was received by p_i

for every $p_j \in A$, $j \neq i$
Check whether the j^{th} entry of BM[m.b] either (i) has '+' or (ii) is blank and there exists B', B'>B such that the j^{th} entry of BM[B'] has '+'

3.0 Conclusions

The TimedCB framework has been introduced to handle group communication in hybrid systems, with synchronous and asynchronous characteristics. The same algorithms and information structure can be instantiated in distinct system models (synchronous, asynchronous, or a hybrid system), which simplifies system design. When a pure synchronous system is considered (g = S), TimedCB can provide early delivery, since logical block completion can be achieved before the pessimistic Δ_{max} bound holds, and also the expiration of Δ_{max} is an accurate indication of failures. When an asynchronous system is considered (g = A), message delivery can only rely on the logical block completion conditions and the bound Δ_{max} can be used to trigger failure suspicions. Moreover, dynamic adaptation can be provided as long as adequate

monitoring mechanisms for updating the sets S and A is available - by using the conditions specified for hybrid systems. Finally, process failures and membership reconfigurations can be realised with the consensus algorithm presented in [8], which is capable of tolerating degraded QoS of components (e.g., from synchronous to asynchronous) and does not require that processes share the same view of system synchrony or QoS at a given time.

References

[1] K. P. Birman, “The process group approach to reliable distributed computing,” Communications of the ACM, vol. 36, no. 12, pp. 37–53, December 1993.
[2] Lamport, L., "Time, clocks, and ordering of events in a distributed system", Commun. ACM, 21, 7 (July 1978), pp. 558-565.
[3] R. A. Macêdo, P. Ezhilchelvan, S. K. Shrivastava, "Modelling Group Communication using Causal Blocks", 5th European Workshop on Dependable Computing, Lisbon, Feb., 1993.
[4] Raimundo A. Macêdo, "Fault-Tolerant Group Communication Protocols for Asynchronous Systems", Ph.D. Thesis, Department of Computing Science, U. of Newcastle upon Tyne, 1994.
[5] Paul Ezhilchelvan, Raimundo A. Macêdo, Santosh K. Shrivastava, "Newtop: a Fault-Tolerant Total Order Multicast Protocol", 15th IEEE International Conference on Distributed Computing Systems, Vancouver-Canada, June 1995.
[6] Raimundo A. Macêdo, Paul Ezhilchelvan, Santosh K. Shrivastava. “Flow Control Schemes for Fault Tolerant Multicast Protocols”. The proc. of the IEEE 1995 Pacific Rim International Symposium on Fault-Tolerant Systems (PRFTS’95), Newport Beach, California. Dec. 1995.
[7] C. Aurrecoechea, A. T. Campbell and L. A. Hauw. “A survey of QoS architectures”, ACM Multimedia Systems Journal, 6(3):138-151, Springer-Verlag, 1998.
[8] Sérgio Gorender, Raimundo José de Araújo Macêdo, Michel Raynal. “An Adaptive Programming Model for Fault-Tolerant Distributed Computing”. In IEEE Transactions on Dependable and Secure Computing Vol. 4, no. 1, pp. 18-31, Jan-Mar, 2007.
[9] M. J. Fischer, N. Lynch and M. S. Paterson. “Impossibility of Distributed Consensus with One Faulty Process”. Journal of the ACM, 32(2):374-382, April 1985.
[10] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki, "Transis: A Communication Sub-System for High Availability", 22nd Int. Symp. on Fault-Tolerant Comp., July 1992.
[11] H. Kopetz and G. Grunsteidl, “TTP – a protocol for fault-tolerant real-time systems,” IEEE Computer, vol. 27, no. 1, pp. 14–23, January 1994.
[12] F. Cristian, B. Dancy, and J. Dehn, “Fault-tolerance in the advanced automation system,” in Proc. Of Fault-Tolerant Computing Symposium, pp. 6–17, June 1990
[13] F. Cristian, “Synchronous atomic broadcast for redundant broadcast channels,” Real-Time Systems, vol. 2, no. 3, pp. 195–212, September 1990.
[14] F. Cristian, “Reaching agreement on processorgroup membership in synchronous distributed systems,” Distributed Computing, vol. 4, pp. 175–187, 1991.
[15] F. Cristian. “Synchronous and Asnchronous Group Communication”, CACM, 39(4), April 1996.
[16] P. Verıssimo and A. Casimiro, “The Timely Computing Base Model and Architecture,” IEEE Trans. Computers, 51(8), pp. 916-930, Aug. 2002.
[17] T.D. Chandra and S. Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems,” J. ACM, vol. 43, no. 2, pp. 225-267,Mar. 1996.
[18] D. Dolev, C. Dwork, and L. Stockmeyer, “On the Minimal Synchronism Needed for Distributed Consensus,” J. ACM, vol. 34, no. 1, pp. 77-97, Jan. 1987.
[19] S. Johnson, F. Janhanian, A. Miyoshi, D. de Niz, and R. Rajkumar. “Contructing Real-Time Group Communication Middleware Using the Resource Kernel”. The 21st Real-Time Symposium Orlando, USA. November 2000.
[20] M. Hiltunen, R. Schlichting, X. Han, M. Cardozo, and R. Das, “Real-Time Dependable Channels: Customizing QoS Attributes for Distributed Systems,” IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 6, pp. 600-612, June 1999.