Large Scale Distribution Support for Cooperative Applications *

François J.N. Cosquer
fjnc@inesc.pt

Paulo Veríssimo
paulov@inesc.pt

INESC †

Abstract

Cooperative applications usually require the ability to reliably disseminate information. Providing large scale support means inevitably, tackling problems such as temporary unreachability, often leading to partition failures. This paper presents a framework targetted at experimenting application-specific partition-processing strategies. One key design issue is to combine application semantics/attributes and/with the services of modern group-based communication subsystems. We show that this approach is promising with the view to providing partition support for distributed synchronised applications.

Keywords: Cooperative work, Group-based systems, Large Scale Distributed Systems, Partition failure.

1 Introduction

The rapid growth of interconnected networks has led to the emergence of a computing infrastucture with enormous potential for hosting a wide range of distributed applications. Amongst the most promising are those oriented toward multi-user collaboration, usually referred to as Computer Supported Cooperative Work (CSCW) or groupware.

Distributed synchronised [1] applications are known to have the most stringent requirements [1]. These include fast response time, handling concurrency control while maintaining consistency across all object instances. Because of the high degree of synchronisation, dedicated functionality is required in order to ease the remote collaboration. This means, for instance, using floor control mechanisms and collaboration awareness modules which, by providing a clear context to the end-users, aim at recreating a meeting-room environment. These modules rely heavily on fast and reliable dissemination of information to all users and thus, put strong requirements on the supporting platform, especially on the communication subsystem.

This raises complex problems since it is known that, in asynchronous message-passing computing environments, it is impossible to differentiate a slow node or communication link from a crashed node [2]. Many solutions have been developed to encompass the problem, known as failure detection, and have proven to be satisfactory in LAN environments. However, in the context of Large Scale Distributed Computing Systems (LSDCS), current solutions are ill-suited because the high variance and the unpredictability of communication delays drastically increase the probability of incorrect failure detection. That is, whereas LSDCS exhibit physical partition failures, inaccurate detection may also lead to virtual partitions.

Surprisingly, having surveyed a number of groupware applications and platforms [3], we found little or no concern for scaling and fault tolerance issues. This is clearly not acceptable if synchronised cooperative applications are to become widely used over large scale. We also have highlighted the relationship between the requirements of synchronised groupware applications and the functionality provided by group-oriented systems. We are developing a group infrastructure for large-scale networks, NavTech [4], which constitutes the support framework for the work presented in this paper.

Most current systems are designed with the "no-partition" assumption or the "primary partition" in mind [5]. We suggest that a weaker approach should be used in order to provide better service to the users. We believe that dynamic response to partition failures

* This work was partially supported by the CEC, through Esprit Project BR 6360 (Broadcast).

† Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6^{o} - 1000 Lisboa - Portugal, Tel.+351-1-310000.

[1] In order to avoid any confusion with the distributed systems community jargon, we will not talk about synchronous or non-synchronous modes but about synchronised and non-synchronised operations when referring to operations performed simultaneously (i.e at the same time) or independently (i.e at different times) by users.

(i.e reconfiguration) is necessary and cannot be fully transparent to the applications programmer.

One of our main objectives is to develop and experiment with partition-processing strategies. Namely, the evaluation of the trade-offs between the extra partition programming burden and support versus the service provided at run-time to the end-users. Another advantage of our approach is to tackle in a unified framework all possible cases of partitions. By avoiding ad-hoc system decision, we eliminate the concept of virtual partitions [2] while still using the powerful Virtually-Synchronous Communication (VSC) environment provided by group-oriented systems.

2 Overall framework

System assumptions

As mentioned before, the system considered is an asynchronous distributed system where processes communicate through the exchange of messages. Processes and communication links may fail by crashing. Furthermore, there are no bounds on processing speed nor on communication delays.

On top of this, the group-based communication subsystem provides a *strong-partial membership service* as defined in [6]. Informally, this means that at any time in the system it is guaranteed that concurrent views are non-intersecting. Virtual Synchronous Communication is provided on top of the membership service.

The cooperative applications are modeled as a set of nodes manipulating a shared workspace. We consider the case of replicated architectures, in which all objects are replicated at each node. Although this is not a strict requirement. Each object during a collaborative session possesses a read and a write quorum. We do not discuss here the issues linked with original assignments of quorums and under which conditions they may change.

System Architecture

The overall objective of this experiment is two fold: firstly, to get an insight into fault-tolerant algorithms for building a partition-resilient generic groupware platform; secondly, to evaluate the suitability of group-based systems to support cooperative applications in LSDCS. This led us to organise our system as shown in Figure 1. By providing mechanisms at

[2] within the limits of the system assumptions presented in section 2.

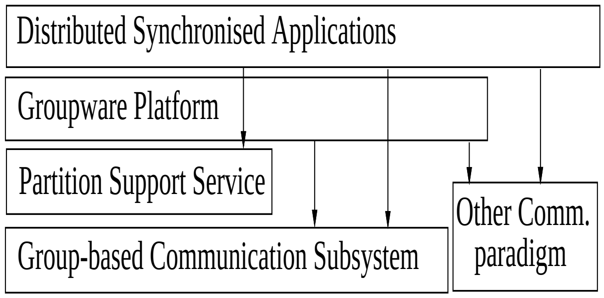

Figure 1: Overall system architecture

an intermediate layer between cooperative applications and the underlying communication subsystem, we allow customisation through policies.

The *Synchronised Applications* are represented at the top and rely mainly on mechanisms implemented in a *Groupware Platform*. This groupware platform layer is intended to hide the complexity of issues such as registration, synchronization of multiple users and information dissemination. By grouping the functionality in a support platform, we free the programmer from the burden of reimplementing generic mechanisms. Dedicated *Partition Support* should greatly ease the task of the groupware platform designer, and thus of the application programmer by providing enhanced functionality. This is the reason why we suggest the *Partition Support Service* layer which provides an intermediate level of abstraction between the *Communication Subsystem* and the *Groupware Platform*.

The shape of the rectangles represented on Figure 1 is illustrating the fact that application requirements may not only rely on the proposed support. This is valid for all the intermediate layers down to the communication subsystem. We foresee that direct use of functionality can be made bypassing intermediate layers. This is represented by vertical arrows. At the lowest level, it is possible to use an alternative to the *Group-based Communication Subsystem* if required. This is represented by the *Other Communication paradigm* box.

3 Design Issues

Synchronised groupware applications exhibit the following attributes:

1. **Involvement of end-users:** This means that unexplained blocking should be avoided. Also, partial and temporary unreachability should not prevent the users from making progress. On the other hand, dealing with users means that it is acceptable and preferable, in some cases, to give the users the possibility to make a decision about

unreachability or about conflicting updates resolution.

2. **Context-free modules:** Synchronized applications have a number of functional modules which do not depend on the structure or on the content of the shared data. This includes floor control policies used for mediating access to shared workspace entities such as audio or video channels, access to shared telepointers over some text or graphical windows, etc. Other modules which are turned towards collaboration awareness and accessed in an interjection-like manner, like shared console windows or subject-based bulletin boards, do not depend on the workspace content either. This is also true for collaboration awareness modules which provide feedback such as activity or state of mind status (i.e idle, unreachable, approving, puzzled, etc.). Once partioned, all these modules should remain accessible following a specified policy.

3. **Complex event scheduling:** These applications require a (possibly) complex agreement phase amongst the users (known as registration phase). We should avoid breaking a complete session because one or a few nodes are unreachable in the system.

4. **Complex start-up procedure:** The shared workspace can be rather sophisticated including various text and graphical windows as well as establishing audio/video channels. It is preferable to maintain a (temporary) partition mode, even with reduced functionality, rather than having to stop and restart a session locally.

As a summary, we should elaborate a strategy that reduces the virtual partition occurrence by delaying crash failure decision while providing support for periods of unreachability.

We propose a two-level approach as described below:

1. Level of Service[3] (LoS): Each node maintains its current state of connectivity through a metrics matrix provided by the output of a specified performability test. This allows the (graphical) collaboration awareness module to visualise the current state of connectivity and possibly keep a record for diagnostic purposes.

2. Partition functionality: Partition runtime information based on roles and weighted voting is used to identify partition types (*main* or *secondary*). It also used to adapt the access rights to shared workspace objects using quorums [7]. In case quorums cannot be reached or if users wish to bypass access rules, a *log service* is provided thus, avoiding blocking. Partition *splitting* and *merging* rules, based on partition type, allow context-free modules to proceed transparently. Upon (complete) reconnection, context-free modules respect original policy while possibly conflicting log updates will be resolved by users as suggested in [8].

[3] We do not overload the term Quality of Service (QoS) often used to describe adaptability to the network and computing resources through reservation and scheduling mechanisms.

4 Partition Support Service

Integration of mechanisms

We propose to integrate application domain specific requirements into our system in the following way:

- Extend Failure Suspector functionality: by adding application-defined performability tests to the mechanisms used to suspect failure and provide a status viewer. This maintains connectivity metrics to be used by higher level modules.
- Extend Membership Services: by allowing user-level decision on unresolved unreachability cases.
- Customise Membership View: by adding extra partition status information to the group membership view. This allows identification of partitions and defines access rights to shared workspace objects.
- Provide a Dedicated Log service: this keeps track of potentially conflicting concurrent updates to the shared workspace during partition failure.

The organisation of the Partition Support Service (PSS) is given in Figure 2. Thick arrows represent the *"depends on"* relationship between the different modules. The PSS is composed of three main modules which we describe below.

The **Partition Level of Service** serves mostly the first level functionality of the PSS. It is organised as 2 sub-modules. First, the *Performance evaluation* provides the mechanisms to specify and run the performability test. This is done through a performability function `pf()` at each node which is usually a `1`$\rightarrow$`N` test. It is a composition of lower level primitives

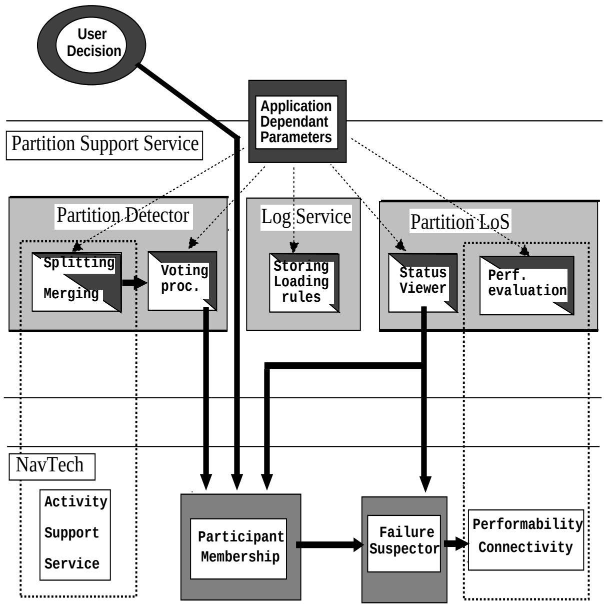

Figure 2: Partition Support Service

measuring parameters like response time, throughput, last-time-heard-of etc. The output of `pf()` is an instant indication of the user-defined application status. The selected status information (status connectivity table `sct`) is managed by the *Status viewer* component and can be consulted by higher level modules when required. Below a certain (user-defined) threshold and upon agreement reached by the membership component the node enters the partition processing mode. Failure suspicion is still a binary decision, however, the *Status viewer* allows the user to eventually follow its evolution or trace it back. Hopefully, this should lead to a better understanding of the situation when a new view is delivered.

The **Partition Detector** serves the second level functionality of the PSS. It is organised as 2 submodules. After a new membership view delivery, the *Voting procedure*, will attempt to first, define the partition type (*main* or *secondary*) and second, calculate objects'access rights quorums. It relies on the notion of roles and weighted voting. Roles, referred to as social roles, are a standard technique used at application level to model the relative importance of users involved in a collaboration. This improves the flexibility of the scheme proposed in [7]. The *main* partition is considered to be unique as in the primary partition model since it was easier to define *splitting* and *merging* rules. However, only further experiments will confirm this choice. We do not consider join and quit operations during partition mode for now. We can easily accept quit operations at UI level, but we delay the system quit up to a safe point. As for joins, we can refuse connection while the application runs in partition mode, for example, or restrict joins to the main only. *Splitting and merging* rules allow consistent assigment of roles and weights throughout the application lifetime.

Finally, **the Log Service** keeps track of potentially conflicting updates to shared workspace data. Each element of the log is structured as a record of the logical update associated with the current view. Reintegrating or replaying a log can be a complex issue. Expecially when tackling concurrent outages resulting in consecutive splits and merges of different partitions of the original view. Further work is underway to partially automate the reintegration phase.

If the system resource limits are reached during partition mode, the user will be asked to make a decision on the log content. We foresee that if such a case happens, it is highly probable that we are in the case of a long lasting partition. The main partition will certainly have made the decision to exclude the unreachable user(s) using the user-level membership decision mechanism. User(s) isolated in a secondary partition can safely save their log on stable storage and quit the application. It might be cheaper after some elapsed time to get an updated version from the main partition instead of trying to run a reconciliation recovery, based on an obsolete view of the shared workspace.

Implementation considerations

The functional blocks in Figure 1 and the final implementation of the Partition Support Service might be embedded partially within the groupware platform and partially inside the NavTech software. For example, the partition detector part can be viewed as an instance of what is usually refered to as *Activity Support Service* in NavTech. This is also valid for the performability evaluation block which is also present in the NavTech architecture. This duality is represented by dotted boxes in Figure 2.

Basic Service Interface

Customised Membership View *CMV [Pri | Aux,* Q_i*,* v_i*]* is defined as *vote(*v_i*, weights, roles)*

Status Metric SM for node i [SM_i]:
is defined as *pf(conn-tests,* n_i*)*

Status Connectivity Table SCT_{i+1}:
is defined as the result of *map pf* v_0 concatenated with the previous table SCT_i.

where:
- v_i denotes the i^{th} view delivered by the system.
- Q_i denotes the quorums in v_i.
- weights and roles are user defined.
- conn-tests is a suite of basic system level connectivity tests provided by the lowest layer.
- map is the standard higher-order function that, given a function and list as arguments, applies the function to each element of the list and returns the new list.

5 Application and experience

We are developing a cooperative graph editor. It uses a shared graph window, including a telepointer associated with a floor policy. The application also runs an audience monitoring module displaying current activity status such as floor owner information and connectivity level. Depending on the use of the tool, for example a brainstorming session, roles such as *main editor*, *co-editor*, *reviewers* and *readers* can be defined with their respective weights.

One of the first actions is to define *pf()* according to the desired performability test. A basic example is to test when it was *last-time-heard-of* and perform the *ping* test every *x* seconds. Alternatives are discussed in [9]. The other parameter to set is the threshold above which a node is said to fail the performability test. We can now see the interest in the connectivity status which will store the result of the performability test. This metric can be displayed as part of the collaboration awareness. In case of partition, it is easy to check if the node is either below the threshold or unreachable.

Using the proposed partition processing support, the programmer defines rules for deciding on partition type. An example is to say that a partition is the *main* if a percentage of the users is with the *main editor*, or alternatively, if a large majority of *reviewers* are in the same view excluding the *main editor* but including the *co-editor*. Any combination is possible as long as only one *main* partition can be identified in the system. We also need to make sure that each partition will elect a temporary floor owner. This is easily done using role precedence. Merging rules will guarantee that rejoining yields a unique floor owner. The net result is that context-free modules allow work to progress just after the partition view is delivered. Furthermore, any access to objects will require the quorum as defined at creation time by its owner. Bypassing the rules is only possible using the Log service.

Using techniques from token-based algorithms [10, 11], we have defined rules for floor control policies and telepointer assigments using the Partition Support Service. This low level support information can be combined with new feedback techniques in order to provide adaptiveness of the application to the system state. Functionality and presentation degradation techniques were applied to the audience monitoring, telepointer and concurrency control modules. Details of these experiments are given in [12].

We believe that this work constitutes a step towards improving collaboration awareness in synchronous large scale CSCW tools and applications. We are now extending this activity in two directions: study the relationship between the failure suspector component and the agreement of membership protocol in the context of those applications; define the requirements for the log service.

6 Conclusions

Initial experiments with partition processing strategies showed the need for:

1. More flexible failure suspector and membership services, namely, ones that can perceive user semantics and be customised to specific situations.

2. Separating application functionality that is context-free (i.e shared workspace data independent) so as to preserve animation in degraded connectivity modes.

Using this approach, the programming burden is low since our PSS only needs a few parameters. Partition identification and quorums techniques based on roles and weights, provide a simple, yet flexible way of capturing application semantics. Our aim is more dedicated towards support for short periods of unreachability rather than disconnected operations. Processor or link crashes rarely hold for long. For example, recent statistics on the T3 backbone show a tendency of 15 minutes or below for outage duration. However, only experience will tell us if this assumption holds.

This initial work confirms our intuition about modern group-oriented systems in providing suitable functionality for supporting cooperating applications over large scale. Using Virtual Synchronous Communication, one can implement relaxed WYSIWIS (What You See Is What I See) paradigm such as WYGIWIG (What You Get Is What I Get) which satisfy users' requirements in many application cases. This seems a promising direction, especially in the light of the efforts developed for providing group-oriented platforms with high configurability [13] or targetted at large-scale environments [14, 15, 16]. However, the proposed approach is at an early stage and further experiments are underway in order to confirm our first results.

Acknowledgments

Carlos Almeida, Manuel Sequeira and Luis Rodrigues provided comments on an earlier draft of this paper, which improved its content and presentation substantially. We are grateful to Malachy Murphy for the careful proofreading of this document.

References

[1] Colin Allison and Mike Livesey. Coping with Concurrency in Real Time Groupware. In *Proceedings of the Usenix SEDMS IV*, pages 289–295, September 1993.

[2] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the Association for Computing Machinery*, 32(2):374–382, April 1985.

[3] François J.N. Cosquer and Paulo Veríssimo. Survey of Selected Groupware Applications and Supporting Platforms. Technical Report RT-21-94, INESC, Rua Alves Redol 9-6°, 1000 Lisboa, Portugal, September 1994. (Also available as BROADCAST Technical Report [2nd year - Vol 1]).

[4] Paulo Veríssimo and Luís Rodrigues. The NavTech Large-Scale Distributed Computing Platform. Technical Report RT-94-xx, Broadcast Project, INESC, Rua Alves Redol 9-6°, 1000 Lisboa, Portugal, 1994. (in preparation).

[5] Aleta Ricciardi, Andre Schiper, and Kenneth Birman. Understanding Partitions and the No Partition Assumption. In *Proceedings of the 4th Workshop on Future Trends of Distributed Computing Systems*, pages 354–360, September 1993.

[6] A. Shiper and A. Ricciardi. Virtually Synchronous Communication based on a weak failure suspector. In *Proceedings of the 23rd Int. Conf. on Fault Tolerant Computing Systems*, June 1993.

[7] David K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the Seventh Symposium on Operating System Principles*, pages 150–162, December 1979.

[8] P. Kumar and M. Satyanarayanan. Log-Based Directory Resolution in the Coda File System. Technical Report CMU-CS-91-164, School of Computer Science, Carnegie Mellon University, Pittsburg, 1991.

[9] François J.N. Cosquer and Paulo Veríssimo. Using Tailored Failure Suspector to Support Distributed Cooperative Applications. Technical Report RT/95, INESC, Rua Alves Redol 9, April 1995. (In preparation).

[10] Shojiro Nishio, Kin L. Li, and Eric G. Manning. A Resilent Mutual Exclusion Algorithm for Computer Networks. *IEEEE Transactions on Parallel and Distributed Systems*, 1(3):344–355, July 1990.

[11] I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. *ACM Trans. Comput. Syst.*, 3(4):344–349, 1985.

[12] François J.N. Cosquer, Pedro Antunes, Nuno Guimarães, and Paulo Veríssimo. Adaptive Synchronous Cooperation over Large Scale Networks. Technical Report RT/95, INESC, Rua Alves Redol 9, March 1995.

[13] R. van Renesse, Ken Birman, Robert Cooper, Brad Glade, and Patrick Stephenson. The Horus System. Technical report, Cornell University, July 1993.

[14] Ozalp Babaoglu and Andre Schiper. On Group Communication in Large-Scale Distributed Systems. In *Proceedings of the 6th ACM-SIGOPS Europe Workshop*, September 1994.

[15] P. Ezhilchelvan, R. Macedo, and S. Shrivastava. Newtop: A fault-tolerant group communication protocol. Technical report, Broadcast Project 2nd Deliverables, October 1994.

[16] Y. Amir, D. Dolev, S. Kramer, and D. Malki. A Communication Sub-System for High Availability. In *Proc. 22nd International Symposium on Fault-Tolerant Computing*, pages 76–84, July 1992.