

A Cluster Cooperation Protocol in Distributed Systems

SungHoon Park, SuChang Yoo, BoKyoung Kim

School of Electrical and Computer Engineering, Chungbuk National Unvi. Korea

E-mail: spark@cbnu.ac.kr, izibt@nate.com, agiboss210@naver.com

Abstract - *In distributed systems, a cluster of computer should continue to do cooperation in order to finish some jobs. In such a system, a cluster membership protocol is especially practical and important elements to provide processes in a cluster of computers with a consistent common knowledge of the membership of the cluster. Whenever a membership change occurs, processes should agree on which of them should do to accomplish an unfinished job or begins a new job. The problem of knowing a stable membership view is very same with the one of agreeing common predicate in a distributed system such as the consensus problem. Based on the termination detection protocol that is traditional one in asynchronous distributed systems, we present the new cluster membership protocol in distributed wired networks.*

Keywords: Synchronous Distributed Systems; Cluster membership; Fault Tolerance; Termination Detection

1 Introduction

In distributed systems, a cluster of computer should continue to do cooperation in order to finish some jobs. A cluster membership protocol is especially helpful tools to allocate processes in a same cluster with a same view of the membership of the cluster. Whenever a membership change occurs, processes can consent to which of them should do to finish a waiting job or begin a new job. The problem of getting a stable membership view is very same with the one of getting common knowledge in a synchronous distributed system such as the consensus problem [1].

The Cluster membership protocol [2] is that every process connected in a network requires getting a stable same cluster membership view if all connected process are belong to just one cluster. The problem was widely discussed at the study community. The reason for this great study is that many distributed systems need a cluster membership protocol [3,4,5,6,7]. In spite of such practically usefulness, to our knowledge there is only a few research that have been committed to this problem in a wired arbitrary connected computing environment.

Depending on process failure and recover, network topologies are changed and process may dynamically connect and disconnect over a wired network. In such wired networks, group membership can be changed so much, making it a special critical module of system software part. In wired arbitrary network systems, a lot of environmental adversities are more common than the static wired network systems such as that can cause loss of messages or data [8]. In particular, a process can easily get to fault by hardware or software problem and disconnect from the wired network. Implementing fault-tolerant distributed applications in such an environment is a complex and difficult behavior [9,10].

In this paper, we propose a new protocol to the cluster membership protocol in a specific wired distributed computing system. Based on the termination detection protocol that is traditional one in asynchronous distributed systems, we address the new cluster membership protocol. We make up of the rest of this paper as follows. In Section 2 we address the system model we use. In Section 3, we describe a specification to the cluster membership problem in a traditional synchronous distributed system. We also address a new protocol to solve the cluster membership problem in a wired arbitrary computing system in Section 4. In Section 5, we address conclude.

2 Computing system model, definition and assumption

In this section, we describe our models for capturing behavior of distributed systems. We use these models for reasoning about correctness of our protocol as well as for analysis of distributed computations. Our model for distributed systems is based on notice passing, and all of protocol is around that concept. Many of these kinds of protocol have analogs in the shared memory world but will not be addressed in this paper.

First, we define our system model based on some assumptions and after that we address our goals. We model a distributed system as a loosely coupled message-passing system without shared memory and a global clock. Our

distributed computation model for a wired network is made up of as an undirected graph. That is, the undirected graph is described as $G = (V, E)$, in which vertices V facing each other with set of distributed process $\{1, 2, \dots, n\}$ ($n > 1$) with unique identifiers and edges E between a pair of process correspond the fact that the two process are in each other's transmission radii. Hence, in our distributed system a channel to directly communicate with each other which changes over time when processes move.

Every process i has a variable N_i , which denotes the neighboring processes, with that i can directly communicate the neighboring processes. Every process communicates with a channel that is bidirectional; $j \in N_i$ iff $i \in N_j$. More accurately, in the network $G = (V, E)$, we decide E such that for all $i \in V$, $(i, j) \in E$ if and only if $i \in N_j$. Depending on process's movement, the graph could be disconnected that means that the network is partitioned. Because the processes may alternate their position, N_i position would be unexpectedly changed and therefore G also may be changed accordingly. The assumptions about the processes, wired network and system architecture are followings.

Every process is distinguished by a unique identifier. The unique identifiers are used to distinguish processes during operating the cluster membership search process. Channels and links are bidirectional that means first n first out, i.e. every process receives notices based on the sequence that are delivered over a link between two neighboring processes. Many topology changes may be arbitrary occurred when the process moves in wired networks. That makes a lot of network partitioning and merging. Processes can make a fault to be crash arbitrarily at random and can recover again at any time.

Without network partition, the sender and the receiver do successful notice delivery that means the notice would be successfully delivered only when the two processes remain connected for the all period of notice transfer. Every process has a big receiving buffer enough to avoid buffer overflow all the time in its lifetime. Even though a finite number of topology changes, every process i eventually has a same view of cluster membership of the cluster to which i belongs.

3 Cluster membership specification

We assume that our specification is as followings, it is consist of four properties for a cluster membership protocol.

Safety(1) : At any time, all processes in the cluster have a stable consistent view.

Progress : If there are no more changes in the each views of the processes in one cluster, they eventually getting to their stable consistent views.

Validity : If all processes in a event know a view as their local view and they have eventually reached their stable states, then the last process of their sequences of global views are all at same position and must be equal to each other.

Safety(2) : When a view is committed as a global view, it

cannot be changed.

The first property describes agreement. Consistent history must be an unchanged one for any program that satisfies the specification. The second property shows termination of global view. When the state and event of all processes are unchanged, the processes are eventually getting to close changing their output results. The third property removes trivial solutions where protocols never getting on any new view or always determine on the consistent view.

4 Cluster membership protocol

At this section, we address a cluster membership protocol that was operated upon the termination detection protocol by scattering computations. After these sections, we will describe in detail the method that this protocol may be accommodated to a distributed system.

4.1 Cluster membership in a wired network

We first address our cluster membership protocol in the wired network settings. In which we assume that process and channels have no faults.

The protocol is made up of three phases running at the process that starts the cluster membership protocol.

1) The first phase that is a diffusing phase and it works by first diffusing the "who" notices.

2) The second phase that is a searching phase and it runs by then accumulating the id of every process that is consist of the wired networks. We represent this computation starting processes as the *start process*.

3) The third phase is a closing phase that is managed by deciding the same view and announcing it as a stable new view to all process.

The start process will have the information enough to decide a uniform cluster membership view after taking all process' ids completely and the start process will then broadcast it to the rest of the process in the network. The three kinds of notice, *Who*, *Ack* and *View* are used to manipulate the operations.

As the first phase is diffusing computing phase, *Who* notice is used to make a start of the cluster membership protocol by diffusing the *Who notice*.

1) The first Phase:

When cluster membership protocol is launched at a start process s , the start process makes a replying queue wl and a accepted queue rl and starts a *scattering computation* by forwarding an *Who notice* to all of its immediate neighboring processes. At the starting point, the replying queue makes up of only its most close neighboring process's ids and the accepted queue has nothing.

When process i receives a *Who notice* from the neighboring process for the first time, it immediately sends the *Ack* notice to the start process and propagates the *Who notice* to all its neighboring process except the process from which it first accepted an *Who notice*.

The *Ack* notice sent by process i to the start process contains the ids of all its neighboring process that are needed for the start process to decide the stable view of the process connected with a distributed network. After that, any *Who*

notice accepted by other neighboring process will be ignored.

2) The Second Phase:

Searching phase. When the start process receives the Ack notice was taken out from the process j , it takes j out from the replying queue and gets j into the accepted queue and as soon as possible it detects sequentially the each process's id included in the Ack notice. If there is the some process in the Ack notice which has already been accepted, i.e. that means it is in the accepted queue, it is dismissed. If it is not in the accepted queue, it is inserted into the replying queue of start process. The start process will be suspends for the Ack notice from one.

The replying queue is increasing and decreasing repeatedly when it was accepted based on the accepted Ack notices, however the replying queue is continually increasing by accepting the Ack notices. But the replying queue at the end could have no element and the replying queue could insert all ids of processes connected to the wired networks whenever the start process accepted the Ack notices from all other processes. Therefore the start process eventually has much information enough to decide the stable view of the cluster based on the replying queue. That is because the replying queue could be eventually unoccupied and it means that the start process has accepted the Ack notices from all the process.

3) The Third Phase:

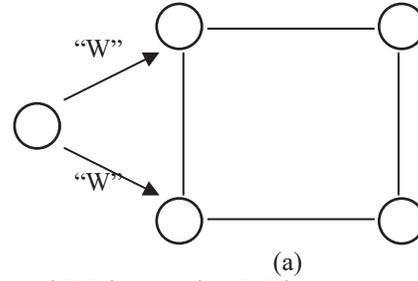
Once the start process has accepted Acks from all other process, it decides the stable view based on the replying queue and forwards a View notice to all other process to let know the current view of the cluster. We show some sample running protocol as the protocol execution to explain more specific features. We address the protocol in synchronous setting even though all the behaviors of the protocol are practically asynchronous. We assume that the network shown in Figure 1(a) is asynchronous. In this shape, and for the all of the paper, thin arrows denote the route of Who notice's move and dotted arrows denotes the way of route of Ack notices to the start process.

As shown in Figure 1, process A is a start process that starts wl_a and rl_b with $\{B,C\}$ and $\{A\}$ at each and starts a scattering computation with forwarding out Who notices (indicated as "E" in the shape) to its immediate neighbors, viz. process B and C, shown in Figure 1(a). As indicated in Figure 1(b), process B and C in turn forward the Who notice to its most close neighbors only except the start process. It sends the Ack notice with close neighboring process queue to the start process A. Hence B and C also send Who notices to each other.

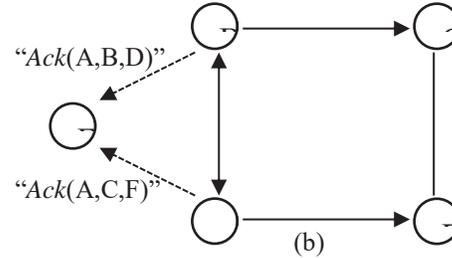
But B and C do not acknowledge to the start process about the Who notices because process B and C have already accepted Who notices from the start process at each. The information of neighboring process is piggybacked upon the Ack notice sent by all process.

Upon hearing Ack notices from B and C, process A renews $wl_a = \{B,C\}$, $rl_b = \{A\}$ with the close neighboring process information piggybacked at the Ack notices.

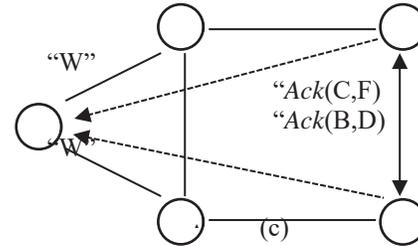
$$wl_a = \{ B,C \}, rl_a = \{ A \}$$



$$wl_a = \{ D,F \}, rl_a = \{ A,B,C \}$$



$$wl_a = \{ \}, rl_a = \{ A,B,C,D,F \}$$



$$wl_a = \{ \}, rl_a = \{ A,B,C,D,F \}$$

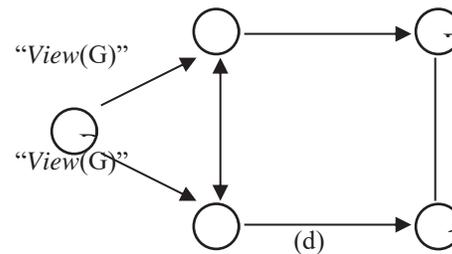


Figure 1: An example of cluster membership protocol execution on the process search protocol.

The Who notices is transmitted over the arrows at the edges and the dotted arrows going parallel with the edges denotes Ack notices. In Figure 1(c), the process D and F also send the Ack notices to the starts process at the time they accepted the Who notice s from the B and C one by one.

Each of these Ack notices includes the ids of the neighbor. All the time, the start A accepts all acknowledgments from all of other process except itself in Figure 1(d) and then determines the stable view between the clusters and forwards it, that is the View notice displayed in Figure 1(d).

5 Conclusion

We have addressed here the study of distributed cluster membership protocol for distributed, wired networks and proved it to be correct based on the symbolic dynamics of finite state machine obtained by linear probability model. We have also shown that the symbolic formal specification of property in our cluster membership protocol based on linear temporal logic is to reason the protocol correctness.

In real world, the wired network topology is actively and lively changing at random and that dynamic network changed configuration causes frequent connection and disconnection of process over the wired network. In spite of weakness about wired networks, our cluster membership protocol specification guarantees the safety and progress property could be always satisfied.

As mentioned in the introduction, our main goal has been to design cluster membership search protocol and prove decidability of consistent view in as simple a fashion as possible without paying much attention at wired networks to the consistent membership view on every process even though complexity issues of distributed networks.

We are however convinced that process search protocol and linear set theory techniques can considerably relax the strong property of safety in many of our constructions. In particular, a more careful logical design of cluster membership protocol for specific classes of wired networks can be performed using the results from our design of convergence properties of consistent cluster membership view. This could lead to a significant improvement of our protocol from a practical environment point.

Finally, in a practical setting one may generalize the cluster membership protocol to more fit in some distributed systems according to network environments. It will be interesting to explore whether safety or progress could be weakened depending on distributed computing environmental factors.

6 References

- [1] Amir Y, Moser E, Melliar-Smith P, Agarwal D, Ciarfella P. The totem single-ring ordering and membership protocol. *ACM Trans. Computer Systems*. 1995; 13(4): 311-342.
- [2] Anceaume E, Charron-Bost B, Minet P, Toueg S. On the formal specification of cluster membership services. Technical Report Computer Science Dept. Cornell Univ. 1995; 95(3): 1534-1559.
- [3] Anker T, Chockler G, Dolev D, and Keidar I. Scalable cluster membership services for novel applications. *Proceedings of Workshop on Networks in Distributed Computing (DIMACS 45)*; 1998 Aug 21-23; P. 23-42.
- [4] Keidar I, Sussman J, Marzullo K, Dolev D. A client server oriented protocol for virtually synchronous cluster membership in WANs. *Proceedings of 20th Int'l Conf.*

Distributed Computing Systems; 2000 April 15-17; 3(1); P. 234-244.

- [5] Brunekreef J, Katoen J, Koymans R, Mauw S. Design and analysis of dynamic leader cluster membership protocols in broadcast networks. *Distributed Computing*. 1996; 9(4): 157-171.
- [6] Bottazi D, Montanari R, Rossi G. A self-organizing cluster management middleware for distributed ad-hoc networks. *Computer Communications*. 2008; 31(13): 3040-304.
- [7] David Powell. Special section on cluster communication. *Communications of the ACM*. 1996; 39(4): 50-97.
- [8] Pradhan D, Krishna P, Vaidya N. Recoverable distributed environments: design and tradeoff analysis. *Proceedings of Annual Symposium on Fault Tolerant Computing*; 1996 Oct 12-15; P. 16-25.
- [9] Briesemeister L, Hommel G. Localized cluster membership service for wired networks. *Proceedings of International Conference on IEEE Parallel Processing Workshops*; 2002 Mar 05-18; P. 94-100.
- [10] Hatzis K, Pentaris G, Spirakis P, Tampakas V, Tan R. Fundamental control protocols in distributed networks. *Proceedings of 11th ACM SPAA*; 1999 Oct 19-21; P. 251-260.