# Brief Announcement: Fast Protocol Transition in A Distributed Environment

Xiaoming Liu and Robbert van Renesse
Computer Science Department, Cornell University
Ithaca, NY 14853, USA
{xliu,rvr}@cs.cornell.edu

Adaptivity is a desired feature of the distributed systems. Because many characteristics of the environment (network topology, active process distribution, *etc.*) may change from time to time, a good system should be able to adapt itself and perform sufficiently well under different conditions.

Modern distributed systems are generally built from a set of components. Such a system has the freedom to adapt itself by switching from using one component to another. Because most components in the distributed systems are running protocols, an agreement must be achieved among the processes when doing the adaptation.

The traditional approach to do the protocol switch is by using the two-phase-commit algorithm, in which a coordinator first broadcasts a "prepare" message, and all the other processes pause their work and send back acknowledgments. Each process is buffering messages from its own application at this point. After the coordinator receives all the acknowledgments, it broadcasts a "switch" message, and upon receiving which all the processes resume working using the new configuration. This approach is clean and easy to implement. However, it has two shortcomings: (1) the reconfiguration is not "smooth", *i.e.*,, the overhead is large; (2) it is not scalable due to the centralized scheme.

We propose a method which allows the protocol switch with very little overhead. It is scalable as well. The method is based on the fact that if two protocols $P_1$ and $P_2$ are derived from the same abstract specification $AS$, there exist converting functions $f$ and $f'$ that can convert the local state of a process in one protocol to another. We can then build a hybrid protocol based on $P_1$, $P_2$, $f$ and $f'$ that can make smooth adaptation at runtime.

We briefly describe the generic algorithm of the hybrid protocol in three steps: (1) One process initiates the protocol switch by broadcasting a "switch" message; (2) When a process learns about the switching, it stops the current protocol by starting buffering application messages. It then sends out its information that other processes may need in order to convert their local states; (3) When a process gets all the needed information, it converts its local state to that of the new protocol using the converting function provided. It then starts working using the new protocol immediately. Each configuration is associated with a timestamp, which is tagged to the messages sent in that configuration. When a message with a timestamp greater than the local timestamp arrives, it gets buffered, and is processed after the local conversion finishes. To ensure that there is only one reconfiguration at any time, a token mechanism is being used. The hybrid protocol is smooth, because the protocol switch is not depend on the slowest process as in the two-phase-commit approach. It is efficient because the local state conversion saves many unnecessary memory operations.

As an example, we apply our algorithm to two types of atomic broadcast protocols, namely, sequencer (S-)protocol and token (T-)protocol. In the S-protocol, each process has a buffer (*Sbuf*) holding the messages yet to be ordered by the sequencer. In the T-protocol, each process has a buffer (*Tbuf*) holding the messages to be broadcast when the token arrives. When switching from S-protocol to T-protocol, the sequencer sends out the information including the number of the messages from each process that have been ordered so far. Other processes convert by transferring the unordered messages from *Sbuf* to *Tbuf*. When switching from T-protocol to S-protocol, the process with the token sends the ordered information to the sequencer, and all the processes will transfer the messages in *Tbuf* to *Sbuf* by sending the message proposals to the sequencer.

We implement the algorithm with our group communication toolkit. The following table shows the performance of the hybrid protocol (*HYB*) versus that of the two-phase-commit protocol (*2pc*). In the test, each process broadcasts 100 messages in each round, when a process receives all the messages in this round, it starts a new round. We switch the protocol every 3 rounds. The result being shown (in *m*sec/round) is the average round latency of 100 rounds for 3 processes. The S-protocol and T-protocol data is of no protocol switch and just for the comparison. The algorithm works much better when the number of processes increases.

| *HYB* | S-protocol | T-protocol | *2pc* |
|---|---|---|---|
| 13.3 | 9.9 | 11.8 | 151 |

Our algorithm provides a generic way of building efficient and scalable adaptive protocols. We believe it is a step towards the modular approach to adding new functionalities, such as adaptation, to the distributed systems.