Apparent Causality for Distributed Termination Detection

Farhad Arbab
email: farhad@cwi.nl
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Termination detection in a system of distributed processes is a classical problem in distributed computing. Distributed Termination Detection (DTD) has been extensively studied in the past twenty years and it is known to be a difficult problem to solve efficiently, because it involves properties of the global state of a distributed system. DTD is an especially important basic form of coordination. Appropriately general DTD algorithms can also be adapted for certain other coordination tasks in parallel and distributed systems, such as barrier synchronization, consensus, etc.

Many DTD algorithms exist and a recent survey of 35 of them [1] introduces a taxonomy and identifies 8 different characteristics for their classification and evaluation. This survey concludes by remarking that "[an] algorithm [with favorable ranking in all 8 dimensions,] if one exists, would be a huge development in this field." As difficult as DTD is in its classical setting, considerations for dynamicity and mobility in a distributed system further complicate the DTD problem and render most existing DTD algorithms non-applicable.

It is customary to assume that the only means of communication among the processes in a distributed system is message passing. Furthermore, it is customary to distinguish between *normal messages* exchanged among the processes in a system, and the *control messages* required by a DTD algorithm. One of the most interesting properties of a DTD algorithm is the maximum number of control messages it requires to detect the termination of a (run of a) system in which a total of n normal messages are exchanged. This measure is sometimes called the *message complexity* of a DTD algorithm. The message complexity of reasonably general DTD algorithms for a system consisting of m processes exchanging n normal messages is typically of the order of $m \times n$. Furthermore, the average performance of such algorithms is not significantly better than their worst case.

We introduce the notion of *Apparent Causality* as a relation among the messages in a system of distributed processes, from which we derive the concepts of message *histories* and *futures*. Apparent causality and message histories are inherently local properties which can be evaluated at the level of each process, whereas message futures are inherently global system-level properties. Histories and futures of messages are examples of histories and futures of more general observables in a distributed system. We propose Back To The Future (BTTF) as a generic method for computing futures from histories, and use this technique to construct three different symmetric algorithms:

1. BTTF Transitory Quiescence (BTTF-TQ) is a generic, efficient algorithm that leads a distributed system to a state containing the distributed knowledge that there are no pending messages;

2. Yet Another Wave Algorithm (YAWA) uses the BTTF technique to implement a generic DTD wave algorithm with certain interesting properties of its own; and

3. BTTF Wave is our main algorithm, which combines BTTF-TQ and YAWA to obtain a general symmetric DTD algorithm that is equally suitable for classical settings as for dynamic systems of distributed mobile processes.

A key concept that underlies all three algorithms is the fact that the real cost of communication in a distributed computing system is essentially a function of the number of transmitted messages. In other words, within "reasonable limits" sending a longer message costs the same as sending a shorter one. Furthermore, the occasion that a message uses the full capacity allowed by those "reasonable limits" for each transmission in a system is indeed very rare. Intuitively, our algorithms take advantage of this fact to reduce the number of control messages they need. Our algorithms utilize the unused capacity that is collectively provided by the transmission of all normal messages in a system to piggy-back and transmit a good part of the control information they need to run.

Apparent causality is the key relation that creates this control information in terms of message histories. The BTTF scheme ensures that message histories reach a select subset of message senders, who use this information to reconstruct the futures of their respective messages. The significance of message futures is in the fact that they reveal certain global properties of a distributed system. The BTTF-TQ algorithm uses message futures to extract (partial) information about quiescence in a distributed system. The YAWA algorithm uses message futures to determine the coverage of the rounds of its token waves. The BTTF Wave algorithm uses message futures for both of these purposes in order to reduce the number of rounds of token waves as compared with YAWA.

Messages A message has a unique identity and contains some information. A message is sent by a single unique process (its producer or sender) to a single unique process (its target or receiver). The most important information contained in a message is its *value*, which is what the sender intends to communicate to the receiver. A message may contain extra information in addition to its value. Specifically, we require each message to also carry its own identifier and (a part of) its own history.

The value of a message is either *normal data* or *control data*. Normal data is the set of all application data values exchanged through messages. Messages that contain normal data as their values are called *normal messages*. Control data is the set of values required by a DTD algorithm to be exchanged through messages. Messages that contain control data as their values are called *control messages*. A distinguished subset of control data is called *token values* and messages that contain such values are called *tokens*.

Apparent Causality Consider the externally observable behavior of a process, P, in a parallel or distributed system. Every message has a unique sender and a unique receiver. We can observe that at certain points in time P actually reads (i.e., consumes) one of the messages it has received, and we can also see that at certain other points in time, P sends messages to one (single-cast) or more (multi-cast) processes in the system. Taking P as a black-box, there is no way for us to know what is the true relationship between any of its input and output messages, or indeed if they are related at all. Nevertheless, we can discern a certain *apparent causal* relationship among the input and output messages of P, expressed in terms of a formal relation called *Apparent Causal Precedence* (ACP).

Intuitively, every time a process P produces a message m_i, we associate a (possibly empty) set of messages, $m_j, \quad j \geq 0$, that P has read during the period ending with the production of m_i, as the apparent causal precedents of m_i. The decisions about the starting point of the relevant period and exactly which m_j messages read during this period are related to m_i as its apparent causal precedents, both depend on the type of m_i. If m_i is a token, then the period starts at the time when P sent its last token, or the creation time of P. The messages that are related to m_i, in this case, are all the token messages read during this period. If m_i is not a token message, then the period starts at the time when P sent its last non-token message, the last time P became passive, or the creation time of P. The messages that are related to m_i, in this case, are all normal messages read during this period.

A message that has no apparent causal precedent is called an *initial message*. A message that is the apparent causal precedent of only non-token control messages is called a *final message*.

Causal Chains A sequence of messages m_i, for $i \geq 1$, such that $m_i \prec m_{i+1}$, is called a *causal chain* (of m_1). A precondition for (proper) termination in a distributed system is that after a certain point in time, no process sends a message. Thus, we are interested only in systems wherein all causal chains are finite.

A message can be the apparent causal precedent of more than one other message. Therefore, a single message can be the head of more than one causal chain of messages. Because a causal chain cannot contain a cycle, the set of all causal chains of all messages in a system forms a directed acyclic graph.

Histories Consider the set W of the maximal length causal chains of all messages in a run of a system that end with a particular message, m. Other causal chains that end with m are shorter, are completely overlapped by the longer chains in W, and contain only redundant information. The set of longest non-empty proper prefixes of chains in W is called the history of the message m. In other words, the history of m consists of all non-empty chains that result from chopping m off the end of all chains in W. Observe that (only) initial messages have empty histories and that every chain in the history of a message always starts with an initial message.

A process that never produces an initial message is called a **reactive process**. A process that produces one or more initial messages is called a **pro-active process**.

Futures Consider the set W of the maximal length causal chains of a particular message, m. Other causal chains of m are shorter, are completely overlapped by the longer chains in W, and contain only redundant information. The set of longest non-empty proper suffixes of chains in W is called the future of the message m. In other words, the future of m consists of all non-empty chains that result from chopping m off the beginning of all chains in W. Observe every chain in the future of a message always ends with a final message.

Back To The Future The future of a message contains interesting global information for DTD. For instance, information about the possibility of pending messages and the coverage of a token wave can be extracted from message futures. Because the futures of the messages of reactive processes are totally subsumed by the futures of the messages of pro-active processes, a DTD algorithm needs to be concerned only with pro-active processes. Furthermore, because the futures of initial messages subsume the futures of non-initial messages, a DTD algorithm needs to be concerned with the futures of initial messages only.

While apparent causality is a local property and message histories can be computed incrementally and locally, the futures of initial messages are not readily available to their senders. Back To The Future is a simple scheme through which the future of an initial message can be derived from the histories of other messages. This scheme works by ensuring that every chain in the future of an initial message ends with a (control) message (if necessary) sent to the sender of that initial message.

Reconstruction of the Future The future of a message is represented as a tree. As the relevant chains in the histories of other messages become available to the sender of an initial message, it dynamically reconstructs the tree representation of the future of that initial message by *grafting* those chains onto this tree. Trees that represent complete futures are easily recognizable in this scheme. When no pro-active process has an incomplete future tree, there can be no pending messages in the system. When the future tree of an initial token is complete, the token wave initiated by that token has covered all reachable processes and its round is complete.

Implementation and Analysis The implementation of the above-mentioned three algorithms provides an library of interface functions for the computation layer of a distributed application. The primary functions in this library support (multi-cast) send and receive operations for inter-process communication. All aspects of the DTD protocol are transparently handled by these high-level functions which isolate the application code from the details of the DTD algorithm.

The BTTF-TQ algorithm is best regarded as a more control-message-efficient replacement for the commonly used explicit acknowledgment scheme. In fact, in a distributed system where n messages are exchanged asynchronously and their order is not preserved, reaching a state wherein (transient) quiescence is detectable requires n control messages in the worst case. In the same setup, BTTF-TQ is an optimal algorithm that requires only n_c control messages, where $0 \leq n_c \leq \frac{f}{l} \times n$, $0 < \frac{f}{l} \leq 1$, $\log f \leq \frac{\log n}{l-1}$, f is the average frequency of a message in history chains, and l is the average length of a history chain in a final message.

While not intended as a general DTD algorithm, in some special cases (that are not so uncommon in practice) BTTF-TQ is sufficient to detect termination. In these special cases, the message complexity of BTTF-TQ is dramatically superior to other DTD algorithms.

Because the YAWA algorithm relies only on reconstructed (token) message futures to determine the progression of its rounds of token waves, it is independent of any assumptions about the topology of a distributed system. Specifically, this topology can even undergo drastic dynamic changes (e.g., due to mobility) without affecting the execution of YAWA.

The BTTF Wave algorithm ranks quite favorably in the characterization scheme of [1]. Furthermore, it is generic and is suitable for dynamic and mobile systems at no extra cost. The message complexity of this algorithm is $n_c \leq \frac{f}{l} \times n + 2 \times n_r$, where f and l are as above and n_r is the number of wave rounds, which in the worst case, is of the order of m^2 for a system of m processes.

1 References

[1] J. Matocha and T. Camp, "A taxonomy of distributed termination detection algorithms," *The Journal of Systems and Software*, pp. 207–221, 1998.