On the Complexity of the Minimum and Maximum Global Snapshot Problems

L.B. Chen and I.C. Wu

Department of Computer Science and Information Engineering
National Chiao Tung University
1001 Ta-Hsueh Road, Hsinchu, Taiwan

Abstract

Deriving the minimum and maximum global snapshots is very useful for some error detection problems in distributed programs. Several researchers, e.g., Groselj, Chen and Wu, have shown that the minimum and maximum global snapshot problems are linear-time reducible to the maximum constant-ratio network flow (MCNF) problem, here defined as the well-known maximum network flow problem with $m = \Theta(n)$, where m is the number of edges and n is the number of vertices in the given flow network. In this paper we show in a reverse way that the MCNF problem is also linear-time reducible to these global snapshot problems. Thus, we can conclude that the global snapshot problems are "as difficult as" the MCNF problem in terms of time complexity.

1 Introduction

Error debugging and detection is very important in maintaining distributed programs. Our past experiences of implementing a large distributed system (for load balancing) in [8] suggest that a distributed program usually needs rules to follow in order to run correctly. For example, in a distributed program, there may be a number of tokens distributed among processors (e.g., tokens representing numbers of resources and critical sections) and the numbers of these tokens remain constant or bounded within ranges in any snapshot, no matter how the tokens are moved among different processors. The most common way to detect whether the above situation holds is to derive the minimum or maximum number of tokens for all possible snapshots. This is called the minimum or maximum global snapshot problem [6].

Recently, Groselj [6] proposed an interesting method for deriving the minimum global snapshot, which reduces the minimum global snapshot problem to a maximum constant-ratio network flow (MCNF) problem, here defined as the well-known maximum network flow problem with $m = \Theta(n)$, where m is the number of edges and n is the number of vertices in the given flow network. Garg *et al.* [2] independently obtained a similar result. Later, Chen and Wu [3] proposed a general technique, called *normalization*, for deriving both minimum and maximum global snapshots, they showed that these snapshot problems are linear-time reducible to the MCNF problem.

These research results only show that the time complexities of the minimum and maximum global snapshot problems will not be higher than that of the MCNF problem. However, we still wonder whether the time complexities for global snapshot problems can be lower than that for the MCNF problem. In order to resolve this question, this paper shows in a reverse way that the MCNF problem is also linear-time reducible to these global snapshot problems. Thus, we can conclude that the above global snapshot problems are "as difficult as" the MCNF problem in terms of time complexity.

The remainder of this paper is organized as follows. In Section 2, we describe our model and the notation used in this paper. Section 3 and Section 4 discuss, respectively, the time complexities of the minimum and maximum global snapshot problems. Finally, we give a brief discussion in Section 5.

2 Model and Notation

A distributed program consists of *processes* communicating via a network. These processes share no memory and no global clock. Pairs of processes need to communicate via network *channels*. The state of such a program is distributed over these processes and channels in each snapshot.

Events

The states of processes and channels change only when *events* [7], atomic actions, are executed. There are three kinds of events on each processor P that we are concerned with:

- **Internal event**: does a local computation. It may change the state of process P.
- **Send event**: sends a message from process P to another via a channel. It may also change the state of process P.
- **Receive event**: receives a message from another process via a channel. It may also change the state of process P.

Note that each process must start with an *initial internal event* and end with a *final internal event.*

In order to define the chronological order of events, we define that the event e_i *happens before* event e_j, denoted by $e_i \rightarrow e_j$, if and only if one of the following conditions holds [1]: (1) Events e_i and e_j occur in the same process and e_i occurs before e_j. (2) Event e_i is the sent event of a message and event e_j is the receive event of the same message. (3) There exists another event e_k such that $e_i \rightarrow e_k$ and $e_k \rightarrow e_j$.

Global Snapshot

Consider one possible run of a distributed program. The system can proceed from one state to another by executing events chronologically. A set of events E_C is said to be *consistent* if for all events e in E_C, all the events e', with $e' \rightarrow e$, are also in E_C. A *global snapshot* is a collection of states in all processes and channels after executing only a consistent set of events. In this paper, we are only concerned with the total number of tokens, so a *state* value actually can be represented by its token number. The minimum (maximum) global snapshot is that global snapshot showing the minimum (maximum) number of tokens among all possible snapshots.

Event Graph

For simplicity of discussion, we can use an *event graph* to represent one run of a distributed program as follows. (1) A vertex denotes an event. (2) If event $e_i \rightarrow e_j$ and there exists no event e_k such that $e_i \rightarrow e_k$ and $e_k \rightarrow e_j$, there is a corresponding *arc*, denoted by $\langle e_i, e_j \rangle$, from e_i's vertex to e_j's. An arc $\langle e_i, e_j \rangle$ is called a *message arc* if it corresponds to an in-transit message from event e_i to e_j. Otherwise, an arc is called an *internal arc* because it corresponds to an internal state transition inside a process. Clearly, each process has an *internal path* from the vertex of its initial internal event to the vertex of its final internal event without going through any message arcs. An event graph is illustrated in Figure 1. In an internal path, the arc from the initial event is called the *i-arc* and the arc to the final event is called the *f-arc*. An initial internal event is called an *i-vertex* and a final internal event is called an *f-vertex*. For each message arc a, the number of message tokens is denoted by S_a; for each internal arc $a = \langle e_i, e_j \rangle$, the token number of the corresponding process is denoted by S_a, after execution of the event e_i and before the event e_j.

Cuts

A cut in an event graph H partitions the vertices into two disjoint sets such that one, called the *source part* and denoted by V_s, contains all the i-vertices, and the other, called the *sink part* and denoted by V_t, contains all the f-vertices. Thus, it is trivial to see that a cut has at least one arc in each internal path. The cost of a cut is defined as follows:

$$\sum_{\forall a: a=(u,v), u \in V_s, v \in V_t} S_a$$

For example, in Figure 1, the costs of cuts C_1, C_2 and C_3 are respectively 14, 12 and 16. The *minimum (maximum) cut* is the cut with the least (largest) cost among all cuts. The least (largest) cost is called the minimum (maximum) cut cost.

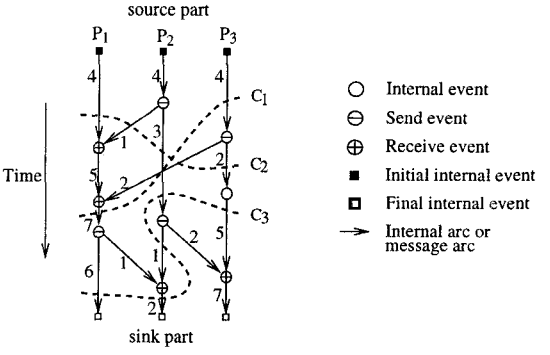

Figure 1: Event graph of a run of a distributed program.

A cut of the event graph is *consistent* if and only if the set of all events corresponding to vertices in the source part is consistent. From this definition, it is obvious that for each consistent cut C, each arc in C must be from the source part to the sink part. This implies the cost of a consistent cut is actually the number of tokens in the corresponding global snapshot.

The minimum (maximum) consistent cut is the consistent cut with the least (largest) cost among all consistent cuts. The minimum (maximum) consistent cut cost is the cost of the minimum (maximum) consistent cut. Clearly, the minimum (maximum) consistent cut corresponds to the minimum (maximum) global snapshot.

In our model, we assume that the event graph with arc costs is given in advance. Garg [4] suggested that in practice, for each event each process sends its token number to a process, called the *checker process*, that runs an algorithm for deriving the minimum or maximum global snapshot.

3 The Minimum Global Snapshot

In this section, we show that the MCNF problem is linear-time reducible to the minimum global snapshot problem. Given a flow network N (see Definition 3.1), the linear-time reduction algorithm given below constructs an event graph H. An example of the reduction is shown in Figure 2. Theorem 3.1 proves that the minimum consistent cut cost of H equals the min-cut capacity in N.

Definition 3.1 *A flow network $N = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a non-negative capacity $c(u, v) \geq 0$. One node s is designated as the source and another node t is designated as the sink. A cut is a set of arcs all incident to two disjoint vertex sets partitioned from V, where one set with node s is called the source set, and the other with node t is called the sink set. The capacity of a cut is the total capacity of all arcs (on the cut) from the source set to the sink set. A minimum cut of a flow network*

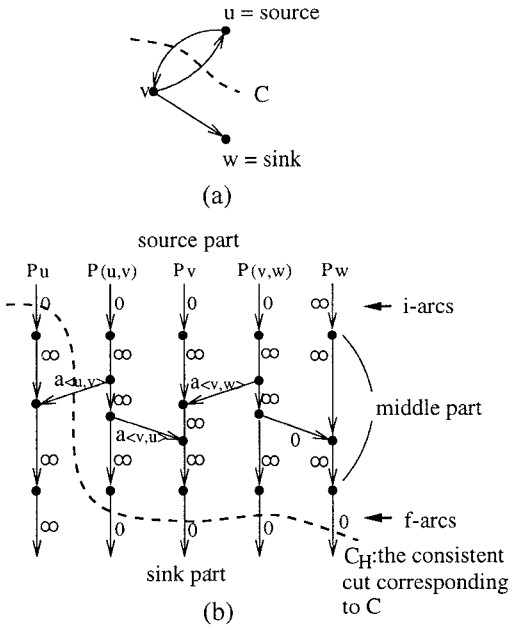

Figure 2: (a) A common graph. (b) The event graph translated from the graph in (a).

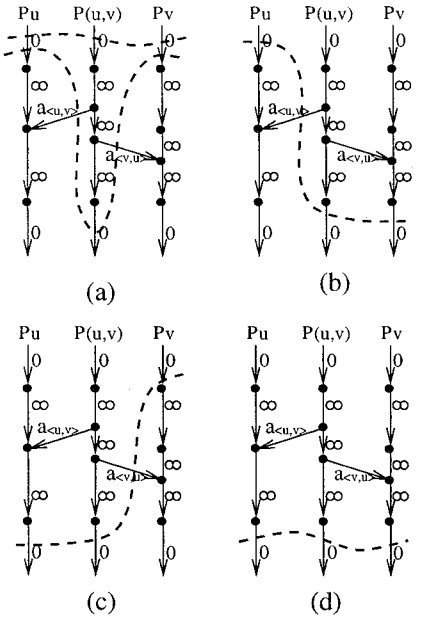

Figure 3: Four cases of a consistent cut cuts the paths P_u and P_v.

is the cut with the least capacity. The least capacity is also called the min-cut capacity.

Reduction Algorithm

1. For each vertex v in N, create the corresponding internal path P_v.
2. For each pair of vertices (u, v), if $\langle u, v\rangle$ or $\langle v, u\rangle$ is in N, create the corresponding internal path $P_{(u,v)}$ (or $P_{(v,u)}$).
3. For each path, add two internal events such that the path is divided into three parts, the i-arc, the middle part, and the f-arc. All message arcs added below must have their vertices incident to the middle parts of paths.
4. For each internal path $P_{(u,v)}$, create a message arc $a_{(u,v)}$ from path $P_{(u,v)}$ to P_u and a message arc $a_{(v,u)}$ from $P_{(u,v)}$ to P_v.
5. Set the cost of f-arc of P_s and i-arc of P_t to ∞, where s is the source and t is the sink. Set the costs of other i-arcs and f-arcs to 0. Set the costs of all other internal arcs to ∞.
6. Set the cost of $a_{(u,v)}$ to the cost of $\langle u, v\rangle$ for each message arc $a_{(u,v)}$, if $\langle u, v\rangle$ is in N, otherwise, set its cost to 0.

Theorem 3.1 *For a given flow network, its min-cut capacity equals the minimum consistent cut cost in the event graph constructed from the above reduction algorithm.*

Proof. The key relationship between the given flow network N and the constructed event graph H is: each consistent cut C_H without ∞ cost in H is said to *correspond* to a cut C_N in the flow network N if and only if the following property holds,

- For each vertex v in N, v is in the source (sink) part of a cut C_N, if and only if the consistent cut C_H cuts across the i-arc (f-arc) of the internal path P_v.

Note that it is possible for more than one consistent cut to correspond to one cut in N, as illustrated in Figure 3 (a).

It therefore suffices to prove that the cut capacity for each cut C_N (in N) equals the minimum cut cost among all consistent cuts (in H) corresponding to C_N. The reason is explained as follows. Since all the consistent cuts without corresponding cuts in N must cut across the middle part of some path (the proof is omitted), their cut costs will be ∞ (see Step 5 of the above algorithm). Therefore, the minimum consistent cut in H is also the minimum capacity among all cuts in N, that is, the min-cut capacity in N.

We now prove that the capacity of each cut C_N (in N) equals the minimum cost among all consistent cuts (in H) corresponding to C_N. For each pair of vertices (u, v), if there exists $\langle u, v\rangle$ or $\langle v, u\rangle$ in N, we have the following four cases.

1. Both vertices u and v are in the source part. In N, edges between u and v will contribute no capacity. In H, all the consistent cut C_H in H corresponding to the cut C_N in N must cut across the i-arcs of internal paths P_u and P_v. In this

case, as illustrated in Figure 3 (a), the consistent cut C_H will cut across either the i-arc or the f-arc in $P_{(u,v)}$ (ignore the middle part because of its ∞ cost, as mentioned above). For the former, none of the message arcs $a_{\langle u,v\rangle}$ and $a_{\langle v,u\rangle}$ will be cut, and therefore, the added cost is zero. The costs of the above two arcs must be added to the latter. Therefore, we must choose the cut across the i-arc of $P_{(u,v)}$ for the minimum consistent cut, we must choose the one across the i-arc of $P_{(u,v)}$, because it adds no cost to the cut cost.

2. Vertex u is in the source part and v in the sink part. In N, the cost of $\langle u,v\rangle$ will be added to the capacity. In H, the consistent cut C_H corresponding to the cut C_N in N must cut across the i-arc of P_u and the f-arc of P_v. According to the definition of consistent cut, C_H must cut across the f-arc of $P_{(u,v)}$, as illustrated in Figure 3 (b), and therefore must also cut across $a_{\langle u,v\rangle}$. Thus, the cost of $a_{\langle u,v\rangle}$ (i.e., the cost of $\langle u,v\rangle$) will be added to the consistent cut cost.

3. Vertex u is in the sink part and v in the source part. For reasons similar to those above, the cost of $\langle u,v\rangle$ in N will be added to the capacity. In H, the same cost will also be added to the consistent cut cost, as illustrated in Figure 3 (c).

4. Both vertices u and v are in the sink part. In N, edges between u and v contribute no capacity. In H, a consistent cut C_H cuts across the f-arcs of both P_u and P_v, and therefore also cuts across the f-arc of $P_{(u,v)}$. As illustrated in Figure 3 (d), C_H cuts across neither $a_{\langle u,v\rangle}$ nor $a_{\langle v,u\rangle}$.

Thus, for each pair (u,v), the cost added to the capacity of the cut C_N is the same amount of the cost added to the minimum consistent cut cost among those consistent cuts corresponding to the cut C_N. Therefore, we can conclude that the capacity of the cut C_N equals the minimum consistent cut costs among those consistent cuts corresponding to the cut C_N. □

4 The Maximum Global Snapshot

In [3], Chen and Wu reduced the maximum global snapshot problem to the minimum global snapshot problem using a technique, called *normalization* that enabled them to reduce an event graph H to another event graph H' in which the cost of each consistent cut is exactly the same as that for the original graph and the cost of each *message* arc is zero (for details, see [3]). Thus, they could easily reduce the maximum global snapshot problem to the minimum global snapshot problem by changing the cost S of each internal arc to $M-S$, where M is the maximum cost among all arcs. (Note that in reducing the problem to the maximum flow network problem, we also have to make all internal arcs non-negative. This is shown in [3], but omitted here.)

In fact, we can also easily reduce the minimum global snapshot problem to the maximum global snapshot problem in the same way, i.e., by changing the cost S of each internal arc to $M - S$, where M is the maximum cost among all arcs. Therefore, both minimum and maximum global snapshot problems are linear-time reducible to each other. That is, we can conclude that the global snapshot problems are also "as difficult as" the maximum network flow problem in terms of time complexity.

5 Discussion

Several researchers, e.g., Groselj, Chen and Wu, have shown that the minimum and maximum global snapshot problems are linear-time reducible to the MCNF problem. Since the time complexity for the most efficient MCNF algorithm [5] is $O(n^2 \log n)$, the minimum and maximum global snapshot can also be solved in time $O(n^2 \log n)$.

In this paper we show in a reverse way that the MCNF problem is also linear-time reducible to these global snapshot problems. Thus, we can conclude that the global snapshot problems are "as difficult as" the MCNF problem in terms of time complexity. Since $O(n^2 \log n)$ has been the best time complexity for the MCNF problem for many years, it seems also a difficult open task to improve upon the $O(n^2 \log n)$ global snapshot algorithms (as well as that for the MCNF problem).

References

[1] K.M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, February 1985.

[2] C.M. Chase and V.K. Garg. Efficient detection of restricted classes of global predicates. In *The 9th International Workshop on Distributed Algorithms*, September 1995.

[3] L.B. Chen and I.C. Wu. On detection of bounded global predicates. In *Proceedings of the International Conference on Distributed Systems,Software Engineering, and Database Systems.* Taipei, 1996.

[4] V.K. Garg and B. Waldecker. Detection of weak unstable predicates in distributed programs. *IEEE Tran. Parallel and Distributed Systems*, 5(3):299–307, March 1994.

[5] A.V. Goldberg and R.E. Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM*, 35(4):921–940, October 1988.

[6] B. Groselj. Bounded and minimum global snapshots. *IEEE Parallel and Distributed Technology*, pages 72–83, November 1993.

[7] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communication of the ACM*, 21(7):558–565, July 1978.

[8] I.C. Wu. *Multilist Scheduling: A New Parallel Programming Model.* PhD thesis, School of Computer Science, Carnegie Mellon University, July 1993.