Distribution Systems Efficiency

Omar Al-Heyasat (corresponding author)
Computer Engineering Department, Al-Balqa' Applied University
omarheyasat@hotmail.com, ashraf.abuain@fet.edu.jo, P.O.Box: 15008 Amman 11134
Ashraf Abdel-Karim Abu-Ein
Computer Engineering Department, Al-Balqa' Applied University
omarheyasat@hotmail.com, ashraf.abuain@fet.edu.jo, P.O.Box: 15008 Amman 11134
Hazem (Moh'd Said) Hatamleh
Computer Science Department, Al-Balqa' Applied University, Ajlun University College
Ahmed A.M Sharadqeh
Computer Engineering Department, Al-Balqa' Applied University,
sharadqh_78@yahoo.com
As'ad Mahmoud Alnaser
Computer Science Department, Al-Balqa' Applied University,
Ajlun University College
Jordan, hazim-hh@bau.edu.jo, asad1_99@yahoo.com

Abstract
This paper constructing a measurement system for determining the efficiency of the distributed system. This efficiency depends on many parameters like number of terminals, number of customers and kind of information, number of nodes, and number of messages. It is found that the efficiency depends strongly on number of messages sent or received inside the distributed system, as the number of messages increased the efficiency decreased.
Keywords: distributed system, terminals, efficiency.

1. Introduction

Distributed systems are a mean to decrease the time needed to perform some task and software errors arise from a number of different problems, such as incorrect or incomplete specifications, coding errors, and faults and failures in the hardware, operating system or network. Model checking is an important technology which is finding increasing use as a means of reducing software errors. Unfortunately, despite impressive recent advances, the size of systems for which model checking is feasible remains rather limited. This weakness is particularly critical in the context of distributed systems: concurrency and asynchrony result in inherent non-determinism that significantly increases the number of states to be analyzed and this affects the distributed systems efficiency. As a result, most system builders continue to use testing as the major means to identify bugs in their implementations. There are, however, two problems with software testing. First, testing is generally done in an *ad hoc* manner: the software developer must hand translate the requirements into specific dynamic checks on the program state. Second, test coverage is often rather limited, covering only some execution paths. To mitigate the first problem, software often includes dynamic checks on a system's state in order to identify problems at run-time. Recently, there has been some interest in run-time verification and monitoring techniques, which provide a little more rigor in testing by automatically synthesizing monitors from formal specifications. These monitors may then be deployed off-line for debugging or on-line for dynamically checking that safety properties are not being violated during system execution.

Unfortunately, testing or runtime monitoring of distributed systems involves considerable overhead: for every event (sending of a message, receiving of a message, or a local state update), each process sends a message about the event to a central monitor. The central monitor constructs and analyzes a computation lattice of the global states out of the collected events. Passing messages to a central monitor at every event and constructing a global computation lattice, which can be exponential in size in the number of events, leads to severe communication and computation overhead. The distributed systems may be effectively monitored at runtime against formally specified safety requirements by *distributing the task of monitoring* among the processes involved in the distributed computation. Effective monitoring means not only linear efficiency, but also decentralized monitoring where few or no additional messages need to be passed for monitoring purposes.

Many researches dealt with such problem, Santhi B et al. (2010) presented novel algorithms for energy efficient

scheduling of Directed Acyclic Graph (DAG) based applications on Dynamic Voltage Scaling (DVS) enabled systems. Experimental results show that our proposed algorithms give better results than the existing algorithms.
Byung-G, 2006, considered replication strategies for storage systems that aggregate the disks of many nodes spread over the Internet. Maintaining replication in such systems can be prohibitively expensive, since every transient network or host failure could potentially lead to copying a server's worth of data over the Internet to maintain replication levels. They also names the following insights in designing an efficient replication algorithm emerge from the paper's analysis. First, durability can be provided separately from availability; the former is less expensive to ensure and a more useful goal for many wide-area applications. Second, the focus of a durability algorithm must be to create new copies of data objects faster than permanent disk failures destroy the objects; careful choice of policies for what nodes should hold what data can decrease repair time. Third, increasing the number of replicas of each data object does not help a system tolerate a higher disk failure probability, but does help tolerate bursts of failures. Finally, ensuring that the system makes use of replicas that recover after temporary failure is critical to efficiency. Based on these insights, the paper proposes the Carbonite replication algorithm for keeping data durable at a low cost. A simulation of Carbonite storing 1 TB of data over a 365 day trace of Planet Lab activity shows that Carbonite is able to keep all data durable and uses 44% more network traffic than a hypothetical system that only responds to permanent failures. In comparison, Total Recall and D-Hash require almost a factor of two more network traffic than this hypothetical system.
Koushik S., et al. 2004, described an efficient decentralized algorithm to monitor the execution of a distributed program in order to check for violations of safety properties. The monitoring is based on formulas written in PT-DTL, a variant of past time linear temporal logic that we define. PT-DTL is suitable for expressing temporal properties of distributed systems. Specifically, the formulas of PT-DTL are relative to a particular process and are interpreted over a projection of the trace of global states that represents what that process is aware of. A formula relative to one process may refer to the local states of other processes through remote expressions and remote formulas. In order to correctly evaluate remote expressions, they introduced the notion of knowledge vector and provide an algorithm which keeps a process aware of other processes' local states, if those states may affect the validity of a monitored PT-DTL formula. Both the logic and the monitoring algorithm are illustrated through a number of examples. Finally, they described an implementation of the algorithm in a tool called DIANA.

1.1 Distributed systems

A distributed system is a collection of k processes (p1, . . . , pk), each with its own local state. The local state of a process is given by the values bound to its variables. Note that there are no global or shared variables. Processes communicate with each other using asynchronous messages whose order of arrival is indeterminate. The computation of each process is abstractly modeled by a set of *events*, and a distributed computation is specified by a partial order on the events. There are three types of events:

(1) internal events change the local state of a process;

(2) send events occur when a process sends a message to another process; and

(3) receive events occur when a message is received by a process.

Figure 1 represents a simulation to some distributed system.

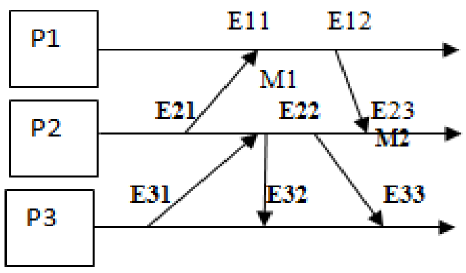

Figure 1 simulation of a distributed system

The time, T, needed for some distributed system to perform some task can be calculated as:

$T= f(T\{NO.(P, E, M)\})$ (1)

where: No. number of, P: number of processes, E: no. of nodes, M: number of messages.

2. Results and discussion

The time needed to perform some given task for a distributed system is adopted to measure the efficiency of such system. From equation (1) the time needed to perform some task is given as:

$T=T_1+T_2+T_3+T_4$ (2)

Where: T_1: time needed for the event to processed internally (inside the process), T_2: the time needed to send a message M and/or receiving it. T_3: the time needed to process the task inside the received node or process, and T_4: the time needed to accomplish the task and display the result.

But as we said the total time needed depends strongly on the construction of the distributed system, i. e. on the number of processes, P, nodes, N, and events or messages, M, so equation (2) can be rewritten as:

$T= PT_1+2MT_2+ NT_3+T_4$ (3)

Now to measure the efficiency of the studied distributed system let the desired time to perform the task is 900µs then the efficiency, η, can be defined as:

η=*actual time/ desired time* (4)

The number of processes here are assumed to be three, with 150 µs each, the time needed to send or receive a messages is 50µs, the time needed to perform the task inside the node is 100 µs, the time to accomplish the task is 100 µs. So the efficiency is η=900/1300= 70%.

It can be noticed that the efficiency depends on number of messages sent or received inside the distributed system. See figure 2 below.

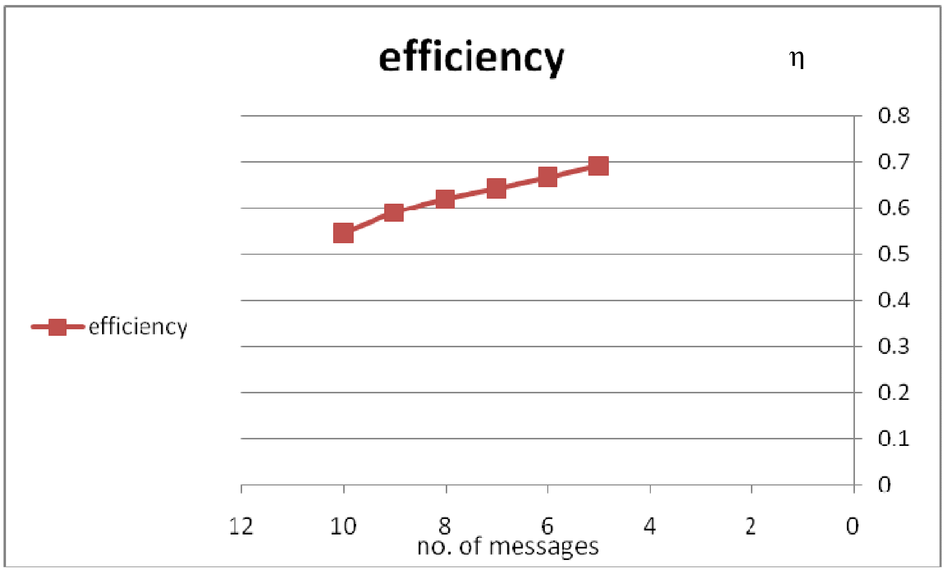

Figure 2 efficiency of the distributed system vs. no. of messages.

3. Conclusion

It is clear that the efficiency of any distributed system depends strongly on the system composition, number of processes, number of events or nodes, no of messages, time needed to process the task, and the time to accomplish the task, as the number of such things increased the time needed becomes longer and so the efficiency decreases.

References

[1] Santhi Baskaran1 and P. Thambidurai, 2010, Energy Efficient Real-Time Scheduling in Distributed Systems, IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 4, May 2010, 35 ISSN (Online): 1694-0784, ISSN (Print): 1694-0814

[2] M. Frans Kaashoek,? John Kubiatowicz,. and Robert Morris, 2010, Efficient Replica Maintenance for Distributed Storage Systems MIT Computer Science and Artificial Intelligence Laboratory, Rice University/MPI-SWS, . University of California, Berkeley, This research was supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660, http:// project-iris.net/. Andreas Haeberlen was supported in part by the Max Planck Society. Emil Sit was supported in part by the Cambridge-MIT Institute. Hakim Weatherspoon was supported by an Intel Foundation PhD Fellowship.

[3] Koushik Sen, Abhay Vardhan, Gul Agha, Grigore Ros¸2004, Efficient Decentralized Monitoring of Safety in Distributed Systems, ACM Journal Name, Vol. V, No. N, October 2004, Pages 1–18.

[4] James D Thomas, Katia Sycara , 2010, Heterogeneity, Stability, and Efficiency in Distributed, jthomas+@cs.cmu.edu, katia+@ri.cmu.edu Sponsored by NSF grant IRI9612131 and ONR grant N000149611222.

[5] Tad Hogg and Bernardo A. Huberman. Controlling chaos in distributed systems. IEEE Trans. on Systems, Man and Cybernetics, 21(6):1325–1332, November/ December 1991.

[6] J. O. Kephart, Tad Hogg, and Bernardo A. Huberman. Dynamics of computational ecosystems. Physical Review A, 40, 1989.

[7] William H. Press, Saul A. Tukolsky,William T. Vertterling, and Brian P. Flannery. Numerical Recipies in C. Cambridge University Press, 1992.
[8] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: A study in multiagent learning. Journal of Artificial Intelligence Research, (2):475–500, 1995.