

Off-line Synchronization of Distributed Logs in Fault Injection Test Campaigns *

Joana M. F. Trindade, Gabriela Jacques-Silva,
Roberto Jung Drebes, Taisy Silva Weber, Ingrid Jansch-Pôrto
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – 90501-970 Porto Alegre, RS, Brazil
{jmfrindade,gjsilva,drebes,taisy,ingrid}@inf.ufrgs.br

Abstract

Developers of network applications face the problem of tracing unexpected events and their consequences when they analyze multiple asynchronous logs generated during a test campaign. Each machine that executes a process in a distributed network application records data in a local log, stamping it with timing information based on its own clock. It is difficult or even impossible to establish the order of events in different logs without a common time line. To maintain the clock synchronization during a test campaign some solutions that change the system clocks can be considered as for example run-time communication protocols, NTP or GPS. Each of these solutions presents some drawbacks. We have developed a tool called LOrd that employs an alternative strategy that allows off-line timestamp synchronization of multiple logs without changing the clocks. LOrd aims to synchronize test data generated in fault injection campaigns.

1. Introduction

Generation of logs is a well-known and established strategy that allows the developers to trace abnormal executions of their software and refine the development. Failure information extracted from recorded errors is being largely used to evaluate the operational reliability of machines or services and to improve system dependability [12].

To debug a system, the analysis of the logs generated during test campaigns can give valuable information about its behavior in certain special faulty conditions. Unfortunately, a common sequential analysis cannot be easily applied to multiple logs generated by a distributed application in different sites.

Each machine that executes a process in a distributed application generates a local log, storing timestamps accord-

ing to its own clock. Each log entry associates events with their respective timing information. In conventional distributed systems, local computer clocks are not inherently synchronized. Being physical processes, the clocks naturally differ from each other. They can run at different speed and can also drift from an external accepted reference time.

It is difficult or even impossible to establish the order of events in different logs without a common knowledge of time. Analyzing logs with different time lines can lead to incorrect diagnosis especially concerning unexpected system behavior under faults.

The problem of clock synchronization arises because the machines in a distributed system typically use internal physical clocks as their primary time source. On most systems, these clocks are accurate to within a few seconds per day, but exceptions to the rule are usual. Desktop computers, for example, operate in power-saving modes in which even the clock is slowed down or stopped [4].

To establish a common time line for our purpose we could choose to synchronize the system clocks during a test experiment using one of the common available solutions as for example executing an online clock synchronization protocol. Other possibility is the synchronization of the timestamps of the multiple logs without altering the original hardware clocks. Using this last strategy, called off-line synchronization [2], a global reference clock can be computed using the relative differences between the system clocks. After a test campaign where asynchronous logs are generated, this global reference is used to correct all of the collected logs timestamps.

Off-line synchronization is simple to use and does not impose a substantial performance penalty. We have developed a tool called LOrd that implements this strategy and has been applied to synchronize log data generated in fault injections campaigns. LOrd is a complementary tool for FIONA ([7], [5]), a communication fault injector aimed to test the dependability of Java networked applications. However, LOrd is an independent tool, which can have a general use in other monitoring environments.

*Project developed in collaboration with HP Brazil R&D and CNPq Project ACERTE (#472084/2003-8)

This paper aims to present LOrd. In the following sections we discuss the problem of clock synchronization and then show the LOrd system model. We detail the off-line synchronization algorithm we used and finalize showing a case example where LOrd was applied.

2. Approaches to Synchronize System Clocks

Synchronized system clocks allow ordering events in a distributed environment. But synchronized clocks are inherently absent in time free networked systems as the Internet. Thus, debugging distributed application over time free networks cannot rely on available clock synchronization.

Clocks can be synchronized using software runtime protocols that periodically exchange messages containing time readings. The algorithm executed by each node for synchronizing the logical clocks can be viewed as a clock process invoked at the end of every resynchronization interval. This clock process is responsible for periodically reading the clock values at other nodes and then adjusting the corresponding local clock value [11]. The goal is to keep the clocks as close as possible to real time and to control the tendency of individual clocks to drift. Also, these runtime protocols must have the capacity to overcome failures whereby a clock might drift at an excessive rate or return erroneous values. Several reports about distributed system clock synchronization protocols may be found in the literature ([14], [16], [4]), as they are often used as building blocks for large real-time systems.

In the early 1990s, the global positioning system GPS introduced a pervasive way to obtain timing information from satellite time sources using a radio receiver. Time obtained in this manner is accurate with resolution in microsecond range unless the GPS signal itself is distorted by unusual atmospheric conditions or problems with the antenna used to receive the signal [4]. GPS can also be combined with software synchronization protocols to synchronize the clocks in a network with an external source time. GPS has generated great expectation in the area of clock synchronization; unfortunately nowadays the general use of GPS receivers in any computer to synchronize clocks is not a common practice.

Following the general availability of GPS, NTP (Network Time Protocol) appeared as a popular solution to get time information using the Internet. NTP uses a symmetric architecture in which a distributed subnet of time servers operating in a self-organizing, hierarchical configuration synchronizes local clocks within the subnet and to national time standards via wire, radio, or calibrated atomic clock. NTP can keep clock skew within tens of microseconds in a LAN, and tens of milliseconds in a wide area network with a negligible cost ([10]). They are usually accurate enough to capture most causal relations that happen in practice.

The three solutions discussed above are runtime solutions. It means that the clocks are kept synchronized during the execution of the applications, consuming performance or demanding special resources as a GPS receiver or an operating system configured to execute NTP. Furthermore, the latter would indirectly require root privileges on all nodes assigned to execute NTP (*e.g.*: configuring *ntpd* on Unix systems). When already available for other purposes, synchronized clocks can also be used to generate a log global timeline. Otherwise, due to the high message exchange rate needed to synchronize the clocks, their use penalizes the system performance or changes the environment where the application is intended to run, leading to inconsistent reasoning about the application behavior and performance.

An interesting solution to overcome the workload of clock synchronization during runtime just to keep the logs synchronized is the off-line synchronization of timestamps. This approach can be used to synchronize timestamps without changing the application environment and without imposing restrictions over the network and machines that are available to a test campaign. If some nodes are using NTP or GPS, the off-line synchronization can profit from it without demanding that every node provides those resources.

3. LOrd system model

We assume a time free networked system where each machine runs one or more processes of a distributed application. Additionally a machine can run monitoring and/or fault injection tools that instrument the application processes and generate their own logs. When a test campaign finishes, every asynchronous logs are collected and then off-line synchronized.

LOrd computes a virtual global clock based on the algorithm proposed by E. Maillet and C. Tron [8]. The original purpose of this algorithm was to monitor parallel programs for performance tuning in cluster environments. Clusters are small scale environments. LOrd extends the way this algorithm is applied to allow its use in medium scale systems comprising thousands of machines.

Maillet and Tron suggested an algorithm that runs in two rounds. It defines one computer as the reference node that computes the virtual global clock. We have proposed extending it to a three level hierarchy with a main reference machine in the first layer, site reference machines in the second layer and local machines in the third layer, as shown in Figure 1. Each machine that belongs to the system runs an agent that is responsible to communicate with its parent machine and/or children in the hierarchy. The hierarchical architecture used in LOrd is implemented as part of distributed architecture of FIONA environment. The reason for three levels is the perfect mapping that can be done from the topology of wide area networks.

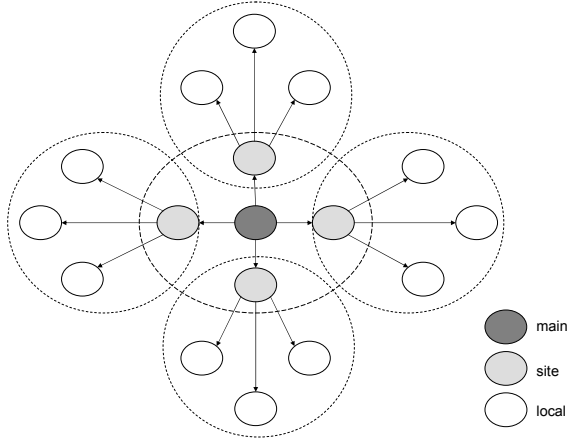


Figure 1. LOrd's three level hierarchy

It must be pointed out that LOrd does not intend to be fault-tolerant. If a machine that computes the virtual global clock crashes, the running test campaign gets lost and must be repeated. It could be a problem if a test campaign consumed a very long time to run but dropping some tests that do not present significant results is an usual adopted approach.

In the next section, we introduce FIONA, the fault injection environment that includes LOrd, then we present the original off-line synchronization algorithm and explain its extension.

3.1. FIONA fault injection environment

FIONA [7] deals specifically with communication faults that disturb UDP messages. FIONA can also launch and coordinate multiple injectors in different nodes over a wide area network to allow emulation of network partitioning [5].

FIONA injects faults in Java programs by instrumenting socket classes using JVMTI (Java Virtual Machine Tool Interface) [15]. JVMTI is a native Java programming interface used by debugging and monitoring tools for Java applications. JVMTI provides both a way to inspect the state and to control the execution of applications running in the Java Virtual Machine.

FIONA injects faults in Java programs instrumenting the datagram socket class. This class provides both `send()` and `receive()` methods for UDP packet service. FIONA is composed of two components: the JVMTI agent, coded in C, and the fault injection classes, coded in Java. When the target runs, the JVMTI agent of FIONA gets notified whenever `java.net.DatagramSocket` is loaded. When notified, the original class image is already loaded in memory, but it has not been processed by the virtual machine

yet. The agent, then, substitutes that image by FIONA's previously instrumented and compiled fault injection class. When the target calls its methods, it actually accesses the instrumented methods. This strategy preserves the integrity of the target source code.

With the aim to distribute a fault injector in every node that forms the distributed test environment, FIONA combines the tree-based architecture proposed by GRM [3] to the protocol strategy used in Ganglia [9]. The communication between the local injectors and the site injector is based on multicast UDP, while it makes use of TCP in the communication between the site injector and the main injector. The choice of local and site injectors can be performed by the user of the tool, based on the number of nodes that participate in a given test campaign.

3.2. Maillet and Tron Synchronization Algorithm

Each process of a M_i node in the distributed test campaign stamps events in a log with its local time lt_i . A node M_r is chosen as a logical clock reference. The method is based on the clock difference between the reference machine M_r and each other machine M_i , $i \neq r$, of the group. The clock difference is measured in two rounds: just before and at the end of a test campaign.

For each node M_i , $i \neq r$, M_r sends a message to M_i . M_i reads its local time t_i and sends this value back to M_r . The local clock of M_r is read before and after receiving this message. The average of these two values is said to be equivalent to the value read in M_i . The results obtained with these measurements are used later to correct the time of the log in each node.

It is possible to model the local time lt_i measured on machine M_i as a linear dependence of the measure of local time in the reference machine M_r , lt_r .

$$lt_i(t) = \alpha_i + \beta_i lt_r(t) + \delta_i \quad (1)$$

In (1) t represents the absolute time, the constant α_i is the offset when t equals zero and the constant β_i is the drift of the physical clock from node i . The more synchronized are two clocks, the more the relative drift is close to one and the relative offset is close to zero. The constant δ_i models granularity and other random perturbations, independent of the time t , and assumed small enough to be ignored. This estimation has good accuracy within relatively extended periods of time (minutes or even hours) assuming that lt_i is a linear function of t .

The algorithm allows to change the local time of M_i registered as timestamps in each log generated in M_i according the virtual global time $LC_i(t)$.

$$LC_i(t) = lt_r(t) \approx \frac{lt_i(t) - \alpha_i}{\beta_i} \quad (2)$$

Note that the initial values of the timestamps and also the drift is corrected in function of the two reading rounds.

3.3. Hierarchical off-line Synchronization Algorithm

Implemented in Java, LOrd computes a virtual global clock using two message exchange rounds as the original Maillet and Tron’s algorithm. The first round runs immediately before each test campaign. The second round runs immediately after the test finishes.

In our extension of Maillet and Tron’s algorithm we have a reference machine M_r for each group. Low-level groups are formed by a site computer and its local computers. Each site computer is the reference machine for its group. The main computer is the reference machine for all site machines which, together with the main computer, form the high-level group.

In LOrd, M_i is either a local node that runs at least one application process during a test campaign and generates at least one log file or it is a site computer. In this last case, the M_r is the main computer. The algorithm is the same for the high-level and low-level groups. It initiates in the main computer. The site nodes answer the main before they act as reference machine in their groups. In the third level, all local computers just play the role of M_i .

The extension implemented by LOrd implies that each site: computes its own site virtual clock according to equations (1) and (2) in its group; correct the logs; and then send them to the main node, where the algorithm is run in the higher level.

The three-level hierarchy allows reducing the time that a single reference node would waste inquiring every other node in the system. In the top level, the *main agent* just needs to inquire the site machines. Each site machine answers the *main agent* and begins to inquire the *local agents* of the machines in its group. The algorithm to correct the time readings of each child node runs simultaneously in all site machines. The hierarchy presents the following advantages: it scales well to thousands of machines in tens of sites, it allows to parallel processing reducing the workload of the main machine and, finally, it diminishes the time to get all the clock readings. This last advantage reduces also the chances that physical parameters of the environment changes during processing time.

3.4. LOrd operation

Figure 2 shows the exchange of messages between the agents in LOrd. The initial phase corresponds to the first

round of the synchronization algorithm. The second round runs after the test execution is finished. The last two phases correspond to log file collection from the machines. The *main agent* triggers the log collection sending a message to the *site agents*, which acknowledge the message and begin to ask for logs in the local machines. The *local agents* are responsible for the file transfer to their parent site computers. Timeouts during the message exchange allow excluding crashed machines.

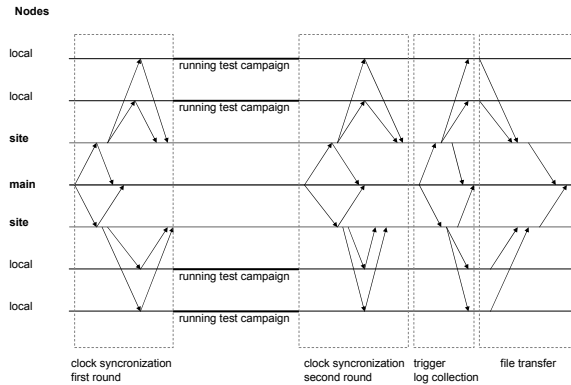


Figure 2. LOrd messages exchange

Each machine that belongs to the system runs a monitoring agent. The *main agent* holds the global virtual clock and synchronizes the timestamps of all log files it receives from the *site agents*. Each *site agent* holds a group virtual clock and synchronizes the log files it receives from its *local agents*. The synchronized log files are merged in the corresponding *site agent* and sent as a single log file to the *main agent*. The *local agent* reads its physical clock and sends it when its *site agent* asks for it. It also sends the log files generated to its *site agent* when inquired. When the *main agent* receives all logs, it uses the global virtual clock to create a common timeline for all logs.

3.5. Case demonstration

In the debugging of distributed applications, where debuggers are not always available or even applicable, the insertion of log statements into the application’s code is a common low-cost alternative. However, in languages such as Java, inserted log statements increase the size of the code, often reducing its speed. Log4j is a logging API (Application Programming Interface) for Java applications aimed to enable logging at runtime without modifying the application binary, so that the log generation does not result in performance costs [1]. With Log4j, the application logging be-

havior can be customized using a proper configuration file. By editing lines in this file it is possible to set properties such as, among others, the logging level and the timestamp format.

In the experiments, we used a Log4j timestamp format `"yyyy-MM-dd hh:mm:ss.SSS"`, based on ISO8061 date format and according to the `java.text.SimpleDateFormat` class symbols. The tests consisted of using FIONA to inject omission faults (*i.e.*, when a process fails to receive or transmit a message on time or at all [13]) in a three node group (hosts `jaguar`, `mercedes` and `buick`) running a UDP application. Also, each node was set to have a different local clock, thus affecting the group clock synchronization.

The host `buick` was chosen as the reference clock in the application of Maillet and Tron's algorithm. FIONA was configured to inject omission faults, with a 20% fault rate, in all the messages sent from `jaguar` to `buick` and `ferrari` under a workload. The workload is a UDP program with a retransmission protocol layer, consisting of a Server (`DrawServidor`) and a Client (`DrawCliente`) programs. A Client instance can only send messages, while Server instances only receive them. Server instances were executed by nodes `buick` and `ferrari` and a Client instance was executed by `jaguar`. During the fault injection experiment, because of the retransmission protocol layer, all affected messages should be retransmitted after a previously specified 50 milliseconds timeout.

In Figure 3, we show the global log after the off-line synchronization (part of the messages normally delivered were omitted). As it is shown, three omission faults were injected, two in the node `ferrari` and one omission fault in the node `buick`. Right after FIONA's log fault records, it is also possible to observe the lost message retransmission event in the application log. Measuring the intervals between these timestamps, we see that the messages were retransmitted within approximately 50 milliseconds from the injected fault, according to the retransmission protocol layer timeout.

4. Related work

NetLogger Toolkit [6] defines a log format and assumes the existence of accurate and synchronized system clocks. The NetLogger Toolkit itself consists of three components: an API and library of functions to simplify the generation of application-level event logs, a set of tools for collecting and sorting log files, and a tool for visualization and analysis of the log files.

Differently from NetLogger, we assume that the application uses any log API as, for example, Log4J or Java™ Logging API. We provide tools for collecting and sorting log files as NetLogger, but we do not rely on syn-

chronized clocks. LOrd aims to solve the problem of the general absence of accurate and synchronized system clocks in conventional network infrastructure such as the Internet.

5. Conclusion

This paper presented LOrd, a tool that performs off-line timestamp synchronization of multiple logs, without changing the nodes hardware clocks. Because of the employed algorithm's off-line nature, LOrd only performs log synchronization before and after the fault injection test campaign's execution. Thus, the chosen method does not substantially penalize the performance of the test campaign. Also, LOrd assures consistency in the historical order of the logs events, thus facilitating the application dependability analysis.

This off-line clock synchronization method is well suited for medium scale distributed systems. Since we do not require a fine-grained time resolution, we infer it can also meet our goals when applied to systems that are largely distributed.

As future work, we propose extending the model to perform n layers synchronization to reduce the distances between node synchronization phases. In this way, and by doing a more precise analysis on the choice of the best reference node, we could increase the accuracy of the algorithm in large-scale distributed systems.

References

- [1] Apache Software Foundation. Log4j, 2005. Available at: [<logging.apache.org/log4j/>](http://logging.apache.org/log4j/).
- [2] P. Ashton. Algorithms for off-line clock synchronization. Technical Report TR COSC 12/952, Department of Computer Sciences University of Canterbury, December 1987.
- [3] Z. Balaton, P. Kacsuk, N. Podhorszki, and F. Vajda. From cluster monitoring to grid monitoring based on GRM. In *Proceedings of Euro-Par 2001*, Manchester, UK, August 2001.
- [4] K. P. Birman. *Building Secure and Reliable Network Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1996.
- [5] J. Gerchman, G. Jacques-Silva, R. J. Drebes, and T. S. Weber. Ambiente distribuído de injeção de falhas de comunicação para teste de aplicações Java de rede. In A. von Staa, editor, *Anais do XIX Simpósio Brasileiro de Engenharia de Software*, Simpósio Brasileiro de Engenharia de Software, SBES, pages 232–246, Rio de Janeiro, 2005. Pontifícia Universidade Católica do Rio de Janeiro.
- [6] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. Netlogger: A toolkit for distributed system performance analysis. In *MASCOTS '00: Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 267, Washington, DC, USA, 2000. IEEE Computer Society.

```

[2005-11-21 02:50:53.361] [buick] Fault injection experiment started
[2005-11-21 02:51:06.951] [jaguar] DrawCliente:74 - Starting application
[2005-11-21 02:51:15.613] [ferrari] Fault injection experiment started
[2005-11-21 02:51:24.933] [jaguar] DrawCliente:24 - Sending msg to buick/143.54.12.175:9000
[2005-11-21 02:51:24.968] [jaguar] DrawCliente:34 - Ack received from /143.54.12.175
[2005-11-21 02:51:24.973] [jaguar] DrawCliente:24 - Sending msg to ferrari/143.54.12.104:8000
[2005-11-21 02:51:24.976] [ferrari] [UdpOmissionFault][SourceIpPort null:0][Receive]
[2005-11-21 02:51:25.026] [jaguar] DrawCliente:43 - Re-sending msg to ferrari/143.54.12.104:8000
[2005-11-21 02:51:25.052] [jaguar] DrawCliente:34 - Ack received from /143.54.12.104
[2005-11-21 02:51:35.297] [jaguar] DrawCliente:24 - Sending msg to buick/143.54.12.175:9000
[2005-11-21 02:51:35.299] [buick] [UdpOmissionFault][SourceIpPort null:0][Receive]
[2005-11-21 02:51:35.349] [jaguar] DrawCliente:43 - Re-sending msg to buick/143.54.12.175:9000
[2005-11-21 02:51:35.351] [jaguar] DrawCliente:34 - Ack received from /143.54.12.175
[2005-11-21 02:51:35.353] [jaguar] DrawCliente:24 - Sending msg to ferrari/143.54.12.104:8000
[2005-11-21 02:51:35.355] [ferrari] [UdpOmissionFault][SourceIpPort null:0][Receive]
[2005-11-21 02:51:35.405] [jaguar] DrawCliente:43 - Re-sending msg to ferrari/143.54.12.104:8000
[2005-11-21 02:51:35.407] [jaguar] DrawCliente:34 - Ack received from /143.54.12.104
[2005-11-21 02:51:35.524] [jaguar] DrawCliente:24 - Sending msg to buick/143.54.12.175:9000
[2005-11-21 02:51:35.529] [jaguar] DrawCliente:34 - Ack received from /143.54.12.175
[2005-11-21 02:51:36.797] [buick] Fault injection experiment finalized
[2005-11-21 02:51:54.231] [ferrari] Fault injection experiment finalized

```

Figure 3. Test Campaign Synchronized Log

- [7] G. Jacques-Silva, R. J. Drebes, J. Gerchman, and T. S. Weber. FIONA: A fault injector for dependability evaluation of Java-based network applications. In *Proc. of the 3rd IEEE Intl. Symposium on Network Computing and Applications*, pages 303–308, Cambridge, MA, 2004. IEEE Computer Society Press.
- [8] E. Maillet and C. Tron. On efficiently implementing global time for performance evaluation on multiprocessor systems. *J. Parallel Distrib. Comput.*, 28(1):84–93, 1995.
- [9] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7), 2004.
- [10] D. L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE Transactions on Networks*, pages 245–254, June 1995.
- [11] P. Ramanathan, K. G. Shin, and R. W. Butler. Fault-tolerant clock synchronization in distributed systems. *IEEE Computer*, 23(10):33–42, October 1990.
- [12] S. Rudraraju. Evaluating web software reliability based on workload and failure data extracted from server logs. *IEEE Trans. Softw. Eng.*, 30(11):754–769, 2004. Member-Jeff Tian and Student Member-Zhao Li.
- [13] F. B. Schneider. What good are models and what models are good? pages 17–26, 1993.
- [14] B. Simons, J. Lundelius-Welch, and N. Lynch. An overview of clock synchronization. In B. Simons and A. Spector, editors, *Fault-Tolerant Distributed Computing*, pages 84–96. Springer Verlag, 1990. (Lecture Notes on Computer Science 448).
- [15] Sun Microsystems. Java Virtual Machine Tool Interface, 2004. Available at: <http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/>.
- [16] P. Verissimo, L. Rodrigues, and A. Casimiro. CesiumSpray: a precise and accurate global clock service for large-scale systems. *J. Real-Time Systems*, 12(3):243–294, 1997.