MSc in Computing (Communications Software)

A Lightweight System for Just-In-Time Aggregation of Machine Generated Data in a Distributed Network

By
Michael
Dreeling

Waterford Institute of Technology

Department of Computing, Mathematics and Physics

January 1, 2013

Table of Contents

1 Introduction

Application Data Management or ADM is a research area concerned with the aggregation, storage and presentation of information typically emitted from Enterprise Applications. ADM is a subset of Enterprise Information Management or EIM [27]. EIM usually joins together Business Intelligence and raw Application Data. Large scale commercial EIM systems started to appear in 2003, and since then many of the larger software vendors such as Tibco, Oracle and SAP [28] have created their own systems. The goal of each system is to present the Machine Generated Data [17] or MGD being emitted from applications and servers. Over the past few years, there has been a focus in EIM on dealing with the challenge of Big-Data as companies such as Amazon, Facebook and Zynga generate terabytes of application information per day. This data, when organized, can be used in the analysis of system problems, user behavior and security threats.

Most EIM software solutions work in either an Active, Passive or Combined operation mode. An Active system requires installation of modules or agents (sometimes called collectors) on servers which are being monitored. The agents usually run with a relatively high level of access and feed information back to a central aggregator. The agents can very efficiently pass the information back to the server and have extremely fast access to the data. A Passive system on the other hand operates entirely remotely and must work harder to retrieve and analyze the data. Typically, file and directory access will need to be granted to the host system by an Administrator over a particular protocol but software is not required to be installed once access has been granted. For the purposes of this dissertation we will focus on the Passive style.

The major challenges for all EIM systems are

1. Accessing and Transmitting Data
2. Indexing Data
3. Searching Data

Data access is the most straight forward issue to solve in both the Passive and Active style as Administrators must be involved in either solution or need to at least provide some level of configuration. Data indexing and Searching however are much more difficult to solve for and as such typically involve mathematical solutions (such as Bloom Filters[22,23] for instance) to resolve. This dissertation will attempt to design a lightweight Passive analysis system which can be used to quickly aggregate and view “short term” data within a system, rather than detailed historical analysis. Such a system could be used to quickly analyze application issues during a high severity event within an Enterprise.

This system would have several advantages over existing software

- Light installation foot print (1 server)
- Low overall network utilization
- Low overall CPU usage on server and monitored host

A secondary goal of the dissertation is to also couple the collection of MGD with Java Virtual Machine (JVM) performance metrics on Java based systems. JVM based performance metrics are much more difficult to analyze and collect than standard application log data but is extremely useful to have alongside general application log data when troubleshooting.

2 The Research Topic

In this section we will outline modern approaches to data aggregation and analysis of Machine Generated Data. This area has seen considerable innovation over the past 10 years and so there are many methods being used. Many of the systems use proprietary technology and granted patents [24]

2.1 Distributed data analysis approaches

In this section we will outline modern approaches to data aggregation and analysis of Machine Generated and JVM based Data.

2.1.1 Distributed Data Collection methods

Information from Enterprise Applications is typically stored in log files on the disks of servers running these applications. The servers may be running a variety of Unix, Linux or Windows based operating systems.

Each server typically stores application logs in a variety of different areas on disk depending on how the application was developed. In addition to the application logs themselves, the software used to run the application may also have logs, and errors can be reported in any one of these files. Logs break down into several categories:

- Be-spoke Application logs (Developer created);
- Off-the-shelf Application Software Logs (Apache, Weblogic, Tomcat);
- Logs generated from the RunTime Environment – for example JVM based logs (In Java environments);
- Logs generated by the Operating system.

In active EIM systems collection of this data is usually performed by Agents and centrally forwarded to a Log Aggregation server where it can be searched. In highly secured environments with change management concerns, a Passive approach is preferable where a centralized logging server pulls data from each server using common protocols and aggregates it afterwards.

2.1.1.1 Active Analysis (Push)

Compared to systems which do not require custom client software installation, Active Analysis systems typically require a large amount of setup,design and configuration, requiring Agents to be installed on the operating system which is running the monitored application. The system must then be configured to read and forward information to a central location where it is collated for presentation. Agent’s usually forward data over a network port (UDP/TCP or HTTPS) to the central aggregator. This method of collection is the very efficient as the agents have direct access to the data under analysis. This method also scales as the number of nodes under management increases. This is due to the fact that the client machines are running some of the required software, and as such the CPU usage required to retrieve the data from them is more evenly distributed.

2.1.1.2 Passive Analysis (Pull)

Passive Analysis systems typically require less setup and design, with more or less the same amount of configuration as Active systems. One or more central servers are installed on the network and they are configured to remotely read the logs of the monitored application. Drawbacks of this approach are that it is less efficient than having direct disk access to the files in question. The reason for this is that active analysis systems can more easily be configured to filter on certain events and feed the central server when these events occur, passive systems must analyze the data once it has been retrieved, and then decide which data to keep. Also, as it uses existing operating system software (such as SSH) to make connections to remote machines, it is also highly dependent on the operating system under analysis.

2.1.2 JVM Based Data Collection

Although it is quite easy to obtain basic information remotely from the JVM (such as CPU or Memory utilization via JMX[25]) this is usually not enough. Typically an enterprise will need to know exactly how a particular piece of code is performing (i.e an Account Transfer transaction). Analysis of this type of data usually needs to be pre-determined, as run-time instrumentation of the code is quite CPU intensive and may slow down the application itself, leading to further issues. Custom JVM monitoring solutions are almost always Active, requiring the attachment of a JVMTI [26] agent to the JVM so that it can be instrumented (JVMPI for Java 4). JVMTI is the newest API used to access the state, and control the execution, of applications running inside of the JVM. JVMPI is a deprecated version of the API used in Java 4. Both of these API's are used by free, and commercial products, to debug and profile the various versions of the JVM. It is very difficult to passively monitor a JVM due to security issues and the relative obscurity of the data. For instance, it is not easy (and sometimes not technically possible) to introspect the Virtual Machine for your business transactions without having first installed software, or at least signaled to the JVM that you wish to do so. Normally this requires administrative access to the J2EE Container (i.e Oracle Weblogic, IBM WebSphere) or physical server which is running your software.

There are several commercial solutions available to monitor a 'live' JVM, including YourKit, HP Diagnostics, JProbe, JProfiler and AppPerfect. Each of these solutions uses a JVM agent to 'latch' onto the running JVM, whereupon it provides a port on which performance information is released, usually to a custom client.

3 Problem Description

As stated in section 2, EIM systems typically fall into two categories of implementation, Active and Passive. One of the problems with an Active system is that it requires significant setup time and continued involvement with a Systems Administration group for upgrade of agents and application of security fixes to same. Some EIM systems do not even allow a remote collection option without first installing software on the host machine.

The other issue with existing EIM systems is that they may not focus on one particular problem but in fact solve for every possible scenario, which sometimes dilutes features which would otherwise be useful.

Existing EIM systems are largely focused on retrieving value from, and analyzing data across, the entire organization, continually. This generates huge amounts of data which may, or may not be useful.

In certain scenarios, only data over a very short time frame is actually useful, and required to solve the problem. Once this data has been obtained and a solution has been determined, its value steadily decreases.

An example Scenario is detailed below.

3.2 IT Crisis Management

A typical IT crisis support scenario requires an engineer to log into and analyze logs from several different sources at once. The data under analysis does not always follow a pattern and the problem may only be apparent to the engineer observing the data. Also the issue in question may involve discovering a pattern that is across several logs, residing at several separate network locations before a solution may be suggested.

Engineers typically have access to these systems for the purposes of deciphering these problems, but there are few tools, if any, which can be easily setup to analyze data using their existing credentials.

The engineer requires visual access to the following information

- Time ordered log information from all sources
- Methods to blend this data together to discover a pattern
- Methods to visualize errors within the data

3.1 Proposed Solution

I propose a system which can quickly attach and siphon data from a system during a crisis management situation and allow an engineer to assess the state of the application and suggest solutions very quickly.

The system would attach itself to machines through standard protocols, (primarily SSH2 on Unix) and make data requests using standard shell commands. This data would be made available to a Rich UI (such as Flex or ExtJS) using a REST API.

The proposed system would have the following characteristics

- Lightweight Client (Web Based Interface on Rails or PHP)
- Lightweight Server (Multi SSH2 client at the core)
- Client would be a live dashboard of log information and performance data
- Applications and their underlying infrastructure and data files are defined in ‘profiles’
- Minimum Indexing Features – not required as data volume is very low (in comparison to commercial offerings)
- No real historical data – Only real-time (plus or minus X hours)
- Crisis Event export features (i.e a Snapshot of what was logged and how systems were performing during the event)
- The system would operate in a ‘standby’ or ‘active’ mode
- Standby mode is where the system is connected via SSH2 to all servers within a particular Applications’ profile but is not pulling data.
- Active mode is where data is being pulled into memory for analysis and indexing, and as such the dashboard can now be connected to view data.
- The system would have large heap (or general memory requirements) but low disk space or archiving requirements (i.e up to 2GB overflow to disk)
- Disk access would only be required if the user shifts to a window which is no longer in memory.

4 Research Questions

1. Is it possible to build a completely passive MGD collector which can scale when centrally connected to a number of servers?

2. What is the impact such a system would have on the monitoring and monitored servers (cpu, memory, installation) and on the network bandwidth between them.

5 Scope of the Research Program

The following software components will be built as part of this dissertation.

1. A Data model and Database to store information about critical application server resources and their locations. This database will be accessed by the server in order to determine server hostnames, logins, data files and associated applications.

2. A Data Siphoning Server (DSS) Component which connects to a Unix server cluster using SSH2 and/or FTP over SSH2. The server will connect via an SSH2 shell and sit on the connection until instructed to retrieve data. Multiple SSH2 sessions will be required, even to one server, in order to provide information on several data streams to the UI. The SSH2 connection will request data using standard Unix commands such as **tail**, **cat** and **more**.

3. A UI to receive data from the DSS. The UI will provide the user with a list of applications to monitor and once selected will stream the pre-defined data streams to the user. The Applications and their resources will need to be setup in the database as a profile first. This may or may not be performed using the UI

4. JVM Based Data collector agent to feed information to UI. Information on monitored applications and their agents would also be stored in the database.

6 Preliminary Literature Review

Due to the fact that Machine Data analysis products have been so commercially successful, there is an abundance of information on the products themselves but few published papers on exact methods used or their efficiency. Most of the documents referenced in this literature review refer to specific components which have been analyzed or developed, which could help in the collection of machine generated data.

6.1 Existing Products [1]

Several products already exist in this space which perform similar tasks to items listed in this proposal, some of the more popular options are outlined below

- GrayLog2 - [http://www.graylog2.org/] – Active (Push) based system (opensource). Requires client software and an MQ system to push.
- OpenTSDB - [http://opentsdb.net/] – Active (Push) based metrics collection server for applications used by StumbleUpon
- Apache Flume - [http://flume.apache.org] – Active (Push) based log aggregation system written in Java.
- Logstash – [http://logstash.net] – Active (Push) based system written in Java. Requires a local agent or rsyslog installed.
- Splunk - [http://www.splunk.com] – Commerical Push based solution which uses collectors installed on each host.

6.1 Machine Generated Data [10, 17]

This article [10], created relatively recently (Dec 2010) describes machine generated data as information “automatically created from a computer process, application, or other machine without the intervention of a human”. This statement is of course partially true, and is acknowledged as such, due to the fact that information generated by processes or machines is inherently tied to the humans which are interacting with it or issuing commands to the system.

It is more accurately described by Monash [17], in a provisional definition, as information “that was produced entirely by machines *OR* data that is more about observing humans than recording their choices”. The article goes on to further describe the categories under which MGD falls, such as the output of network, telemetry and other computer equipment appliances.

The text also categorizes MGD as information which can be, and probably should be, frequently discarded. This statement fits with the purpose of this dissertation, which is to develop a method to quickly review, but not store MGD in a “sliding window” fashion.

6.2 SSH2 Compression [18]

As we are dealing primarily with uncompressed text files for this system, there is an opportunity to increase the performance of text file transfers using SSH2 compression where available. In their paper [18] “Chromium Renderserver: Scalable and Open Source Remote Rendering Infrastructure”, Brain Paul et al describe the effects of SSH2 zlib compression on compressed and pre-compressed image data. The positive results for SSH2 transfer the XWD uncompressed image format are indicative that compression for text log files will improve performance.

6.3 JVM Introspection [2, 7, 8, 12]

Andrew Wilcox [4] provides an excellent overview of the – **javaagent** JVM option and the **java.lang.instrument** API interface. This can be used as the basis for a custom profiler to feed the Data Siphoning component of the system. However this solution will only work for a JVM which is at version 1.5 or above. Documentation on JVMPI [20, 21] is required to code a profiler which is compatible with version 1.4 of the JVM.

7 Contribution to Research Knowledge Anticipated

The main contributions anticipated are primarily software based as outlined below.

1. A high performance lightweight and secure MGD data siphoning server

2. A UI for interpretation of non-historical MGD data from several sources for the purposes of a problem or crisis triage.

8 Anticipated Milestones

The milestones anticipated for this project are outlined below in the following work packages (wp). Some tasks in the work packages are not necessarily sequential and may be worked on out of sequence.

8.1 WP1: Analysis. Expected Completion: 201301

1. Choose and validate transfer protocol as SSH2 (expected duration: 2 days).

2. Choose UI technology (expected duration: 4 days).

3. Choose server side development language and database system (expected duration: 4 days).

8.2 WP2: Design. Expected Completion: 201303

1. Design multi-client SSH data siphoning server. (expected duration: 7 days).

2. Design UI framework for data viewing (expected duration: 10 days).

3. Design database schema to store application profiles (expected duration: 7 days).

4. Design JVM collector agent (expected duration: 10 days).

8.3 WP3: Implementation. Expected Completion: 201306

1. Implement multi-client SSH data siphoning server (expected duration: 30 days)

2. Implement UI (expected duration: 30 days)

3. Implement JVM Collector (expected duration: 25 days)

8.4 WP4: Data Analysis. Expected Completion: 201307

1. Run load tests on multi-node network on Amazon EC2 (>10 nodes) (expected duration: 5 days)

2. Test and Analyze effect of SSH2 compression on data delivery (expected duration: 3 days)

3. Load Test UI (expected duration: 2 days)

8.5 WP5: Final Write Up. Expected Completion: 201308

1. Dissertation write-up. (Expected duration 30 days)

9 Validation Methodology

The goal of this dissertation is to build a system which can be installed on a central server and then used to siphon information from multiple sources during a problem event. In order to ensure that the requirements for such a system have been met the following tests shall be carried out.

9.1 Multi-SSH2 log file general retrieval and display

1. Test that a particular log file can be retrieved from up to 5 sources simultaneously and displayed with the correct timeline in the UI

2. Test that multiple log files (making up the complete set of logs which are defined for the particular application) can be retrieved from up to 5 sources simultaneously and displayed with the correct timeline in the UI

9.2 Multi-SSH2 log file compressed retrieval performance

1. Test the network performance of an uncompressed text file being retrieved via SSH2 by the DS server with compression switched OFF.

2. Test the network performance of an uncompressed text file being retrieved via SSH2 by the DS server with SSH2 compression switched ON.

3. Test above with different levels of compression and/or algorithms.

10 Resource Requirements

10.1 - Hardware

1. Development machine.

2. Deployment and testing will require several machines deployed on AWS or other suitable cloud provider.

10.2 - Software

1. Java 1.4 and 1.5.

2. Database (SQL Server, Mongo, CouchDB or MySQL).

3. Eclipse IDE.

4. Source Control (Private GitHub).

References

[1] *8+ Splunk Alternatives | DevOpsANGLE*. [online] Available at: <http://devopsangle.com/2012/04/19/8-splunk-alternatives/> [Accessed 1 Jan. 2013].

[2] *Build your own profiling tool*. [online] Available at: <http://www.ibm.com/developerworks/library/j-jip/> [Accessed 1 Jan. 2013].

[3] *CouchDB for access log aggregation and analysis - User Primary*. [online] Available at: <http://userprimary.net/posts/2009/06/13/couchdb-for-access-log-aggregation-and-analysis/> [Accessed 1 Jan. 2013].

[4] *CUED SSH2 - how to use it*. [online] Available at: <http://www-h.eng.cam.ac.uk/help/jpmg/SSH2/adv-use.html#compress> [Accessed 1 Jan. 2013].

[5] *Enterprise Information Management Software EIM | Technology | SAP*. [online] Available at: <http://www54.sap.com/solutions/tech/enterprise-information-management.html> [Accessed 1 Jan. 2013].

[6] *Hierarchical In-Network Data Aggregation with Quality Guarantees - Springer*. [online] Available at: <http://link.springer.com/chapter/10.1007%2F978-3-540-24741-8_38?LI=true> [Accessed 1 Jan. 2013].

[7] *Java Concurrency (&c): Profiling with JVMTI/JVMPI, SIGPROF and AsyncGetCallTrace*. [online] Available at: <http://jeremymanson.blogspot.com/2007/05/profiling-with-jvmtijvmpi-sigprof-and.html> [Accessed 1 Jan. 2013].

[8] *Java Tip 92: Use the JVM Profiler Interface for accurate timing - JavaWorld*. [online] Available at: <http://www.javaworld.com/javaworld/javatips/jw-javatip92.html?page=2> [Accessed 1 Jan. 2013].

[9] *JSTOR: The American Economic Review, Vol. 58, No. 4 (Sep., 1968), pp. 773-787*. [online] Available at: <http://www.jstor.org/discover/10.2307/1815532?uid=3739696&uid=2&uid=4&uid=3739256&sid=21101391568783> [Accessed 1 Jan. 2013].

[10] *Machine-generated data - Wikipedia, the free encyclopedia*. [online] Available at: <http://en.wikipedia.org/wiki/Machine-generated_data> [Accessed 1 Jan. 2013].

[11] *Matt Casters on Data Integration » Real-time streaming data aggregation*. [online] Available at:

<http://www.ibridge.be/?p=204> [Accessed 1 Jan. 2013].

[12] *Splunking the JVM (Java Virtual Machine)*. [online] Available at: <http://www.slideshare.net/damiendallimore/splunking-the-jvm-java-virtual-machine#btnNext> [Accessed 1 Jan. 2013].

[13] *syslog - Splunk is fantastically expensive: What are the alternatives? - Server Fault*. [online] Available at: <http://serverfault.com/questions/239401/splunk-is-fantastically-expensive-what-are-the-alternatives> [Accessed 1 Jan. 2013].

[14] *valz - Scalable and robust distributed data aggregation, as easy as logging - Google Project Hosting*. [online] Available at: <http://code.google.com/p/valz/> [Accessed 1 Jan. 2013].

[15] *Wireless sensor network - Wikipedia, the free encyclopedia*. [online] Available at: <http://en.wikipedia.org/wiki/Wireless_sensor_network> [Accessed 1 Jan. 2013].

[16] *Examples of machine-generated data | DBMS 2 : DataBase Management System Services*. [online] Available at: <http://www.dbms2.com/2010/04/08/machine-generated-data-example/> [Accessed 1 Jan. 2013].

[17] *Examples and definition of machine-generated data | DBMS 2 : DataBase Management System Services*. [online] Available at: <http://www.dbms2.com/2010/12/30/examples-and-definition-of-machine-generated-data/> [Accessed 1 Jan. 2013].

[18] Page 10 - *Chromium Renderserver: Scalable and Open Source Remote Rendering Infrastructure [eScholarship]*. [online] Available at: <http://escholarship.ucop.edu/uc/item/08t699fr> [Accessed 2 Jan. 2013].

[19] *JSch - Java Secure Channel*. [online] Available at: <http://www.jcraft.com/jsch/> [Accessed 2 Jan. 2013].

[20] *The Java Virtual Machine Profiler Interface | Dr Dobb’s*. [online] Available at: <http://www.drdobbs.com/jvm/the-java-virtual-machine-profiler-interf/184405722> [Accessed 2 Jan. 2013].

[21] *The Java Virtual Machine Profiler Interface (JVMPI)*. [online] Available at: <http://docs.oracle.com/javase/1.4.2/docs/guide/jvmpi/jvmpi.html> [Accessed 2 Jan. 2013].

[22] *Bloom Filters by Example*. [online] Available at: <http://billmill.org/bloomfilter-tutorial/> [Accessed 4 Jan. 2013].

[23] *View the latest version that the requested documentation is available in. - Splunk Knowledgebase*. [online] Available at:

<http://docs.splunk.com/Special:SpecialLatestDoc?t=Documentation/Splunk/latest/Admin/Bloomfilters> [Accessed 4 Jan. 2013].

[24] *Splunk Breakthrough Recognized with Grant of U.S. Patent for Organizing and Understanding Massive Machine Data*. [online] Available at: <http://www.splunk.com/view/SP-CAAAF8J> [Accessed 4 Jan. 2013].

[25] *Platform Monitoring and Management Using JMX*. [online] Available at: <http://docs.oracle.com/javase/1.5.0/docs/guide/management/agent.html> [Accessed 4 Jan. 2013].

[26] *JDK 5.0 Java Virtual Machine Tool Interface (JVMTI)-related APIs & Developer Guides -- from Sun Microsystems*. [online] Available at: <http://docs.oracle.com/javase/1.5.0/docs/guide/jvmti/> [Accessed 4 Jan. 2013].

[27] *Enterprise information management - Wikipedia, the free encyclopedia*. [online] Available at: <http://en.wikipedia.org/wiki/Enterprise_information_management> [Accessed 4 Jan. 2013].

[28] *Enterprise Information Management Software EIM | Technology | SAP*. [online] Available at: <http://www54.sap.com/solutions/tech/enterprise-information-management.html> [Accessed 4 Jan. 2013].