Chronon: A modern alternative to Log Files

A. The 5 fundamental flows of Log Files

Log files, “Old School”, are a relic from the 1970s, however even today in 2012, IT infrastructure monitoring relies on these old school methods. Even big data processing tools like Hadoop rely on log files as the source of data to process upon.

That said; log files are fundamentally flawed because:

1. They clutter the code and increase development time

Developers manually have to add logging statements to their applications, increasing the total amount of code, and thereby the time and cost required to develop and maintain it.

2. Trying to ‘predict’ errors or data collection events

The fact that one is essentially trying to ‘predict’ errors in code when adding a logging statements makes them fundamentally broken. When a developer puts a logging statement in their code, they have already anticipated an error around it, and thus that code is usually hardened. In real life, errors occur where one least expects it and thus there is almost never a logging statement in the place where the error actually occurred.

The same goes for data collection. Usually if you want to collect data about events in your log files, you have to plan ahead for it and add appropriate logging in your code. However, by the time you come up with the plan to collect a certain piece of data, the event may have already passed you by.
For example, if you want to collect data on how many are trying to sign up for an evaluation license for your product multiple times, by the time you think of putting in logging or adding extra code to collect that data, a huge amount of your visitors would have already gone, resulting in you losing all that important, actionable business data forever!

All of this is compounded by the fact that when putting logging statements in code, developers are required to put a ‘logging level’ like ‘Warning’, ‘Debug’, ‘Fine’, etc. You may have a logging statement in your code but the program may not be running with an appropriate ‘logging level’ to trigger that statement, effectively rendering it useless.

3. Log files contain very sparse information

The information in log files is usually extremely sparse making it almost impossible to find the root cause of an issue or a particular event in your system. Usually one has to go and add additional logging in the code and rerun the system for a long time for the defect or event to reproduce and be caught in the log files. This process can easily take weeks in many cases.

This same principle can be applied to data collection for big data processing; the data would just be gone forever in this case, resulting in an even worse scenario.

4. The larger the log file, the tougher it is to extract information

This is one of those catch 22 situations with log files. In order to extract more information from log files, there is not only extra cost associated with putting logging statements in your code, but now you get a denser log file which you have to parse through all the noise to extract real information from it.
This is why tools like Splunky, Loggly, etc exist, which add immensely to the cost associated with maintaining and parsing log files.

5. Log files need to be ‘parsed’ to extract data

Since large systems are written by multiple development teams, and are composed of multiple components, each may have a different format as to how data is output in their logs.

For example one team may output:

```
Logger.log(“Email delivered successfully”);
```

while the other team might have something like :

```
Logger.log(“Email: Delivery OK”);
```

The person trying to extract information from these log files has to know about the format of these outputs and form parsing statements to extract data from them. These leads to more costs associated with keeping up to date with the log formats of different components in a system.

B. Existing Log Management Solutions

The huge amount of log files present in IT systems has led to expensive 'Log Management' products coming to market, like Splunk, Loggly, SumoLogic, etc. However, none of these systems manage to address the fundamental flaws in log files described above.

They have proven valuable in IT systems so far because apart from log files, there has been no other way to gain any insight at all into the running of applications. However the Achilles heel of other systems lies in the fact that their analysis only extends to the contents of your log files. Thus; you must "predict" what you want and write this into your code. If you forgot to log something in your system, then you cannot analyze it with these tools; and fact is, regardless of however much logging you add, there will always be new and unanticipated data that someone wants to see and analyze.

The use and pricing structure of these systems only adds to the cost of maintaining logs. These systems are priced based on the amount of data that is indexed. In flaw number 4 above, to extract information from your log files you usually need to add more and more logging to your system. Thus there is a direct correlation between the amount of log data that is produced and the increasing cost of purchasing these systems.

However; as stated in flaws 2 and 3, all these expenses may go to waste, since there will always be a need to look at the data you might not have logged. In this case all the huge expense of the hardware and software tools to store and parse log data go to a complete waste of time and money

C. Chronon

Chronon offers an alternative to log files, as a technology more akin to 2012 than 1970.

1. Technology

a) Recorder

Chronon comes with a highly efficient, low overhead 'Recorder' for Java applications that records the entire execution of your app in a very compact recording. The resulting 'Chronon Recording' can be replayed on any computer and allows a developer to jump to any point in time during the execution of a program and examine every single details of it, like the exact stack trace of a threads and the contents of the memory. This allows to analyze *anything* in your system and you are not dependent on someone *hopefully* putting a log statement where it would be needed months later.

Since applications can typically run for weeks/months at a time, the Chronon recorder allows 'splitting/chunking' a recording at pre-specified time intervals from a central dashboard. For example, you can configure a recording to be split every hour, and thus if you want to examine the events at say 2am and 3am, you can just pull out that one single, small recording file for that time period, instead of pulling out a huge week/month long recording. This allows you to easily clear out older recordings.

Since this 'splitting' can be controlled from a single interface in the 'Chronon Recording Server' product, it is much more preferable to log files, where every component of your system can have a different configuration of when the log files are chunked. For example, one application might have a configuration to create a new log file after every 10mb of data is persisted; while other may have it configured to create one every 5 hours. The configuration location and mechanism of these logs may vary too, making it even more difficult to manage.

b) Replayer / Unpacker

The replay/unpack phase of the recording is typically performed on a separate box from where the recording was initially created. In this phase the recording is expanded and all the data indexed so it can be queried extremely fast.

c) Post Execution Logging

Chronon's groundbreaking 'Post Execution Logging' allows you to add a log statement anywhere in your code *after* it has executed and see the results instantly. The logging statements are added through an external GUI and do not clutter up your code. You can access the full state of your program variables in the 'post execution logging' statements, just like you would in an actual log statement in your program.

Since 'post execution logging' is added *after* the program execution, on a Chronon Recording', it does not suffer from any of the fundamental 'traditional logging' flaws mentioned above. You get the exact data you want without having to sort through any noise or any foreign logging formats.

d) Dynamic Dashboards

Dynamic dashboards are a feature of Chronon Recording Server. They allows you to define dashboards that consist of Post Execution Logging statements. The statements are then executed on the recordings of the JVMs that you want to monitor and are constantly updated as new recordings are created. The results of these statements can be seen in both textual and graphical format in the Chronon Recording Server. This allows you to easily monitor any event in your entire data center without the hassle and expense of log files.

2. Cost

Existing Log Management Solutions charge you by the amount of data they index, thus placing a negative synergy between the data you want to produce and the amount you have to pay.

Chronon on the other hand charges on a per jvm basis. There is no limits or extra charges for the amount of data produced, thus making it a much more economical solution.

3. On Demand Data without barriers

To use Chronon, you don't have to change any of your existing IT infrastructure. If you have log statements in your code and existing log management solutions, you can keep them while you try out Chronon. You just need to add the Chronon java agent to the runtime of your jvm and that's it.

It's completely non-disruptive and you can get started in minutes.

Chronon vs Existing Log Management Solutions (Splunk, Loggly, etc)

	Existing Solutions	Chronon
Analysis limited to contents of log files	✗ No help.	✓ No. Chronon Recordings contain the entire execution of your program, so you can analyze anything you want.
Log statement clutter the code and increase development time	✗ No help.	✓ Post Execution Logging renders the need for log statements useless
Trying to 'predict' errors or data collection events	✗ No help.	✓ You can add a Post Execution Logging statement anywhere you want in your code and see the results as if it was actually present in your program. There is no need to 'predict'.
Log files contain very sparse information	✗ No help.	✓ Chronon recordings contain the *entire* execution of your program. There is missed event or data loss.
The larger the log file, the tougher it is to extract information	✗ They do help with indexing and parsing the log files, but the more data you index, the more they charge.	✓ Extract only the exact information you need with accurate Post Execution Logging and Dynamic Dashboards. Chronon doesn't charge by the amount of data indexed so you don't have to worry about amount of data that is produced.
Log files need to be 'parsed' to extract data	✗ While the tools do help with parsing, the fundamental problem that you *have* to parse in the first place remains.	✓ No need to 'parse' data, just produce it in the exact format you want, in the form of Post Execution Logging.
No centralized mechanism to control chunking and location of log files	✗ No help. Every component of a system can have a different configuration as to how and where log files are stored and chunked	✓ Chronon Recording Server gives a single dashboard where you can control the location of recordins for every single jvm in your data center and also manage how often the recordings are chunked or 'split'.

Chronon and Big Data

Existing Big Data systems like Hadoop:

1. Rely on log files as a huge source of data to process
2. Are thus, limited to the contents of the log files and fundamental flaws associated with them, as described in section 1 above.
3. A lot of work is done in parsing log statements from their native to a custom format to extract meaningful data out of them.
4. Full-fledged development budget needed to extract data.
5. Additional cost on top of Log Management systems

With Chronon:

1. Data can be extracted from Chronon recordings instead of sparse log files.
2. No need to separately process log output most of the times, since using Post Execution logging, you put the exact statement and in the exact format you want.
3. No need for a development time budget. Create a dashboard and see data in minutes!
4. Can scale as easily as Hadoop by adding more machines to unpack/index recording data.
5. In most cases, Chronon Recording Server and its remote dashboards will solve most big data needs without the need for systems like Hadoop.
6. In other, more complex processing cases, which Post Execution Logging may not be able to handle, ***Chronon can complement Hadoop***, where the Chronon recording can act as a rich source for Hadoop clusters to process. This is still better than log files, since you are dealing with a more ‘full’ data source instead of a sparse one.

Sample scenario of data extraction:
Chronon vs Other Log Management Solutions

Other Log Management Solutions:

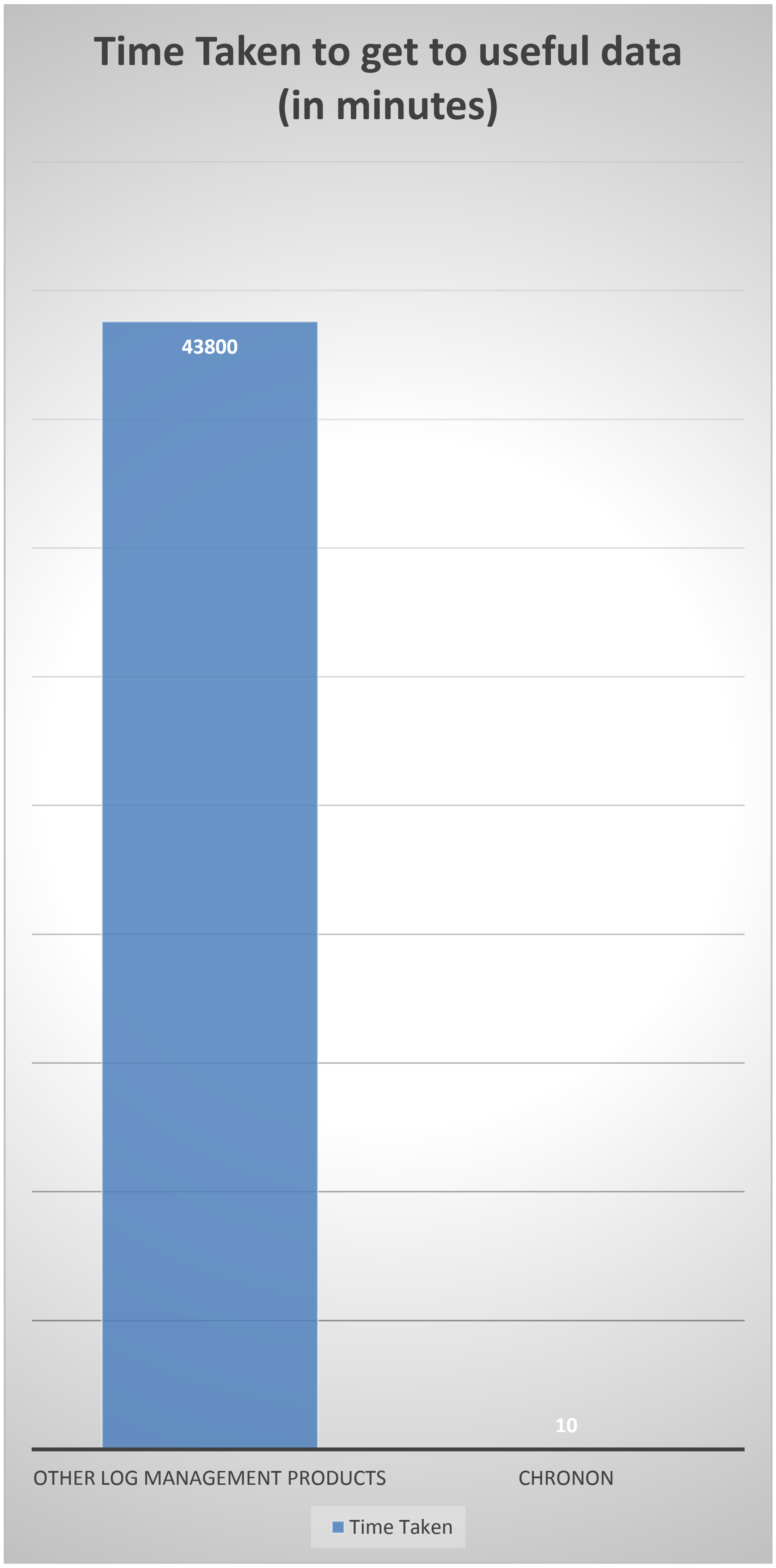
1. Plan data to be collected.
2. Development time and budget allocated to add relevant logging to program to collect data.
3. System is then run for days/weeks to collect data in logs.
4. Development time and budget is assigned to parse log files and collect and analyze log data to extract information from it.
5. If any log statement was missing, or any point of data was missed, restart from Step 1.

Total time taken: *Weeks/Months*
Total Cost: *$$$$$$$$$$*

Chronon:

1. Chronon is enabled on deployed jvms (takes 1-2 minutes).
2. Think of a data you want to analyze.
3. Create a dynamic dashboard with post-execution logging statements (2-5 minutes)
4. See data!

Total time taken: < 10 minutes
Total cost: $


Time Taken to get to useful data (in minutes)
43800
10
OTHER LOG MANAGEMENT PRODUCTS
CHRONON
Time Taken