

Public signup for this instance is **disabled**. Our [Jira Guidelines page](#) explains how to get an account.



Hadoop Common / HADOOP-211

logging improvements for Hadoop

Details

Type:	Improvement	Status:	CLOSED
Priority:	Minor	Resolution:	Fixed
Affects Version/s:	0.2.0	Fix Version/s:	0.3.0
Component/s:	None		
Labels:	None		

Description

Here's a proposal for some **improvements to the way Hadoop does logging**. It advocates 3 broad changes to the way logging is currently done, these being:

- The use of a uniform logging format by all Hadoop subsystems
- The use of Apache commons logging as a facade above an underlying logging framework
- The use of Log4J as the underlying logging framework instead of java.util.logging

This is **largely polishing work**, but it seems like it would **make log analysis and debugging easier in the short term**. In the long term, it would future proof logging to the extent of allowing the logging framework used to change while requiring minimal code change. The propos changes are motivated by the following requirements which we think Hadoops logging should meet:

- Hadoops logs should be **amenable to analysis by tools like grep, sed, awk etc.**
- Log entries should be **clearly annotated with a timestamp and a logging level**
- Log entries should be **traceable to the subsystem** from which they originated
- The **logging implementation should allow log entries to be annotated with source code** location information like classname, methodname, file and line number, without requiring code changes
- It should be possible to change the logging implementation used without having to change thousands of lines of code
- The mapping of loggers to destinations (files, directories, servers etc.) should be specified and modifiable via configuration

Uniform logging format:

All Hadoop logs should have the following structure.

```
<Header>\n
<LogEntry>\n [<Exception>\n]
.
.
.
```

where the header line specifies the format of each log entry. The header line has the format: '# <Fieldname> <Fieldname>...\n'.

The default format of each log entry is: '# Timestamp Level LoggerName Message', where:

- Timestamp is a date and time in the format MM/DD/YYYY:HH:MM:SS
- Level is the logging level (FATAL, WARN, DEBUG, TRACE, etc.)
- LoggerName is the short name of the logging subsystem from which the message originated e.g. fs.FSNamesystem, dfs.Datanode etc.
- Message is the log message produced

Why Apache commons logging and Log4J?



Apache commons logging is a facade meant to be used as a wrapper around an underlying logging implementation. Bridges from Apache commons logging to popular logging implementations (Java logging, Log4J, Avalon etc.) are implemented and available as part of the commons logging distribution. Implementing a bridge to an unsupported implementation is fairly striaightforward

and involves the implementation of subclasses of the commons logging LogFactory and Logger classes. Using Apache commons logging and making all logging calls through it enables us to move to a different logging implementation by simply changing configuration in the best case. Even otherwise, it incurs minimal code churn overhead.

Log4J offers a few benefits over java.util.logging that make it a more desirable choice for the logging back end.


- Configuration Flexibility: The mapping of loggers to destinations (files, sockets etc.) can be completely specified in configuration. It is possible to do this with Java logging as well, however, configuration is a lot more restrictive. For instance, with Java logging all log files must have names derived from the same pattern. For the namenode, log files could be named with the pattern "%h/namenode%u.log" which would put log files in the user.home directory with names like namenode0.log etc. With Log4J it would be possible to configure the namenode to emit log files with different names, say heartbeats.log, namespace.log, clients.log etc. Configuration variables in Log4J can also have the values of system properties embedded in them.
- Takes wrappers into account: Log4J takes into account the possibility that an application may be invoking it via a wrapper, such as Apache commons logging. This is important because logging event objects must be able to infer the context of the logging call such as classname, methodname etc. Inferring context is a relatively expensive operation that involves creating an exception and examining the stack trace to find the frame just before the first frame of the logging framework. It is therefore done lazily only when this information actually needs to be logged. Log4J can be instructed to look for the frame corresponding to the wrapper class, Java logging cannot. In the case of Java logging this means that a) the bridge from Apache commons logging is responsible for inferring the calling context and setting it in the logging event and b) this inference has to be done on every logging call regardless of whether or not it is needed.
- More handy features: Log4J has some handy features that Java logging doesn't. A couple of examples of these:
 - a) Date based rolling of log files
 - b) Format control through configuration. Log4J has a PatternLayout class that can be configured to generate logs with a user specified pattern. The logging format described above can be described as "%d {MM/dd/yyyy:HH:mm:ss} %c {2} %p %m". The format specifiers indicate that each log line should have the date and time followed by the logger name followed by the logging level or priority followed by the application generated message.

▼ Attachments

 acl-log4j.patch	114 kB	31/May/06 19:21
 acl-log4j-ll.patch.tgz	21 kB	01/Jun/06 16:51
 acl-log4j-webapps.patch	1.0 kB	07/Jun/06 17:52
 commons_logging_patch	167 kB	20/May/06 07:05
 patch.txt	15 kB	25/Jul/06 20:57

▼ Activity



▼  [Eric Baldeschwieler](#) added a comment - 12/May/06 13:54

I suggest we use iso8601 time format.


<http://www.cl.cam.ac.uk/~mgk25/iso-time.html>

This would suggest yyyy-MM-ddTHH:mm:ss , such as 2006-05-11T23:47:03


The T is a literal and no one ever likes it. Change it for all I care, but standards are ok. This also suggests UTC, which I think is a good default, but also allows for local time, with a distinct notation 2006-05-11T23:47:03-08. We could support that as a config option if

folks care.

This format is also directly sortable, which is nice and avoids localization issues (MM-dd or dd-MM).


▼  [Doug Cutting](#) added a comment - 13/May/06 00:16

I'm +1 for switching to commons-logging and to log4j by default. But I think we shouldn't mandate a format. It should be possible to embed Hadoop in other systems with other logging standards, and get it to comply with those standards. So most of what you suggest about log formats I think should be couched in the terms "by default", right?


▼  [Owen O'Malley](#) added a comment - 13/May/06 00:40

I'm +1 for using a time format that is sortable. I've been using the sed, awk, grep tools for merging logs files to see trends across time.


commons-logging and log4j sound good.

▼  [Sameer Paranjpye](#) added a comment - 13/May/06 00:49

Yes, the suggestions about formats are meant to be defaults. This is one more reason for using Log4J, it gives you a fair amount of freedom with specifying formats in configuration.

▼  [Doug Cutting](#) added a comment - 13/May/06 01:18

Even log4j should be a default. Hadoop code should only reference the commons-logging api, right? BTW, Sun's logging API also gives you complete freedom in formatting, although you have to write some Java classes, not just configure it with formatting strings as you can with log4j.

▼  [Sameer Paranjpye](#) added a comment - 13/May/06 01:31

Wasn't really thinking in terms of making the logging implementation configurable, but there's no reason that can't be done. Hadoop code won't be invoking any part of log4j or whatever else directly.

Sun's logging does give you complete freedom in formatting, I was pointing out that it's not as flexible as log4j where a lot can be achieved in configuration.


We can have the logging implementation used be specified in configuration. Do you see a lot of people making use of that feature though? My instinct is that they won't...

▼  [Eric Baldeschwieler](#) added a comment - 13/May/06 12:02


What real advantage do we get from all of this flexibility?

One of our goals for attacking the logging system is to allow us to easily process the logs with the system. To do that will require building readers that can deal with the log format and organization. I'd hate to loose that in the interest of complete generality.

Just curious what use case we are after with complete reformability and abstract logging.

▼  [Owen O'Malley](#) added a comment - 17/May/06 23:55

A lot of exceptions are currently being logged at the **info** level and most of them should probably be at the **warn** level. Especially, once we log the level it would be good to find the exceptions by grepping for WARN.

▼  [Doug Cutting](#) added a comment - 18/May/06 00:05


The semantics I use for levels is something like:

SEVERE: if this is a production system, someone should be paged, red lights should flash, etc. Something is definitely wrong and the system is not operating correctly. Intervention is required. This should be used sparingly.


WARN: in a production system, warnings should be propagated & summarized on a central console. If lots are generated then something may be wrong.

INFO, FINE, FINER, etc. are used for debugging. INFO is the level normally logged in production, FINE, FINER, etc. are typically only used when developing.

Is that consistent with the way others use these?

▼  [Milind Barve](#) added a comment - 18/May/06 00:37


At least one place where level should be info and not warn is when we create a file on dfs. It tries to do mkdirs whether directory exists or not. If it already exists, it warns that there is an error creating the directory. I think the warning should be when the directory could not be created because of other factors (such as permissions), otherwise it should be info or even fine.

▼  Barry Kaplan added a comment - 18/May/06 06:48


At the very least switching to commons logging will make it easier when configuring hadoop within other applications. Currently this is one of the few libraries I use that I can't configure to use my standard log4j settings.

▼  Eric Baldeschwieler added a comment - 18/May/06 09:58

True, user errors that cause no harm don't deserve warnings in a central log. The trick is to propagate the issue back to the user...

▼  Barry Kaplan added a comment - 20/May/06 07:05

I needed to get this working with log4j so I quickly ripped out the java.util.logging and replaced it with apache commons. The part I am unsure of is how much configuration of logging do you wish to do from hadoop itself. My opinion is let people configure the log4j the way they want it and hadoop should only choose whatever is available, and log to console if nothing is there (essentially what apache-commons does).

▼  Sanjay Dahiya added a comment - 25/May/06 02:29

On default Log4J configuration format -

Some points -

1. Logging caller class information like - originating method/line no, File name are known to be expensive operations. Also they may not be supported by All JVMs. Do we want these in default config ?
2. For namenode I added an option %X {client}, this enables logging client identification along with msg but needs the code to supply that information using `MDC.put("client", clientName)`
3. We could do a separate logger per client but thats a bad idea as there will be too many loggers.
4. We can have logger hierarchy based on packages/classes and/or some logical hierarchy like - namenode.block.allocation, namenode.block.removal etc. Any comments on what type of log hierarchy you would like to see for the modules you worked on ?
5. We can use MDC for categorizing logs on some criterion other than clients - like blocks, file system. Anythng you would like to see here ?

A log4J format to start with - default is a RollingFile (based on size, can make it based on time also). Pls let me know your requirements if any and I will keep updating this file, once this done migrate codebase to log4j.

remove DEBUG if not needed.

log4j.rootLogger=DEBUG, RFA

log4j.threshold=ALL

Rolling File appender configuration

log4j.appender.RFA=org.apache.log4j.RollingFileAppender

log4j.appender.RFA.File=\${HADDOP_HOME}/hadoop.log

change file size

log4j.appender.RFA.MaxFileSize=1MB

log4j.appender.RFA.MaxBackupIndex=10

log4j.appender.RFA.layout=org.apache.log4j.PatternLayout

log4j.appender.RFA.layout.ConversionPattern=%d{ISO8601} %-5p %c{2} (%F:%M(%L)) - %m%n

logically seperated logger configurations

log4j.logger.namenode=RFA


log4j.appender.RFA.layout=org.apache.log4j.PatternLayout

log4j.appender.RFA.layout.ConversionPattern=%d{ISO8601} %-5p %X{client}

%c

{2}

(%F:%M(%L)) - %m%n

▼  Arun Murthy added a comment - 31/May/06 19:21

Here's attached the patch for switching to commons logging with log4j as the logging framework.

Notes:

- a) Hopefully this can be incorporated asap into hadoop svn since any further commits will entail us to patch anew.
- b) The conf/log4j.properties file uses DailyRollingFileAppender which rolls over at midnight. Another choice is to use RollingFileAppender which will rollover every 1MB and maintain 30 backups, which is currently commented out.
- c) There are some parts of code for e.g. those configuring the log-file, log-levels etc. in the code which we have had to comment out since they aren't supported by Commons Logging. They should now be configured via the .properties file.
- d) The framework as in the patch creates 4 separate logfiles i.e.
 - > \$HADOOP_HOME/logs/namenode.log
 - > \$HADOOP_HOME/logs/datanode.log
 - > \$HADOOP_HOME/logs/jobtracker.log
 - > \$HADOOP_HOME/logs/tasktracker.log


This is done by passing -Dhadoop.log.file=<logfilename>.log via the bin/hadoop startup script and referenced in log4j.properties.

thanks,
Arun

PS: This will have to be committed **before** the libhdfs patch, and after that I will go ahead and create a 2 line patch for TestDFSIO.java (switch java.util.logging to commons logging).

▼  Doug Cutting added a comment - 01/Jun/06 04:01


Unfortunately this patch no longer applies cleanly. Can you please update your source tree and re-generate this patch? I try to process patches in the order they are submitted, but frequently there are conflicts in the queue. Thanks!

▼  Doug Cutting added a comment - 01/Jun/06 04:06

Also, we should not yet remove LogFormatter, but only deprecate it, as user code (e.g., Nutch) may use this class. A good test for back-compatibility of changes to Hadoop's public APIs is to check that Nutch still compiles and runs correctly with an updated Hadoop jar.


If we deprecate LogFormatter in 0.3 then we can remove it in Hadoop 0.4, but we should always give folks at least one release to remove dependencies on deprecated features.

Thanks again!

▼  Doug Cutting added a comment - 01/Jun/06 04:13

One more thing: please don't just comment out obsolete code; delete it. Thanks.

<http://wiki.apache.org/lucene-hadoop/HowToContribute>

▼  Arun Murthy added a comment - 01/Jun/06 16:51

Doug,


Please find the patch for commons-logging/log4j against the latest snapshot.

I have incorporated all your comments i.e. kept LogFormatter.java, removed dead-wood etc.

Please try and apply this patch on a priority basis since it touches quite a bit of the code-base and potentially any commit will necessitate another patch! 😊


thanks,
Arun

PS: We have removed hadoop/conf from the build-time classpath in build.xml since we don't want hadoop's log4j.properties to be picked up by the build-tools. We feel hadoop/conf isn't necessary for building at all. Please correct us if we are wrong.

▼  Arun Murthy added a comment - 01/Jun/06 18:25

Minor point: we will need commons-logging-1.0.4.jar and log4j-1...jar in lib/


thanks,
Arun

▼  Doug Cutting added a comment - 02/Jun/06 05:05


I can apply this by reverting to revision 410692, then use 'svn up' to merge in subsequent changes. But I'm still having trouble building and running unit tests (at least on Windows, which I'm using today, since I'm on the road). The change to build.xml causes hadoop-default.xml not to be found. When I fix that, Jasper fails to be able to compile the jsp pages, since it uses log4j. 'ant clean test' reports:

```
Buildfile: build.xml
[taskdef] log4j:ERROR setFile(null,true) call failed.
[taskdef] java.io.FileNotFoundException: \ (The system cannot find the path s
ecified)
[taskdef] at java.io.FileOutputStream.openAppend(Native Method)
[taskdef] at java.io.FileOutputStream.<init>(FileOutputStream.java:177)
[taskdef] at java.io.FileOutputStream.<init>(FileOutputStream.java:102)
[taskdef] at org.apache.log4j.FileAppender.setFile(FileAppender.java:289)
```

So I'm (sadly) not quite able to commit this yet.

▼  Doug Cutting added a comment - 03/Jun/06 00:48


Okay, I've worked out the configuration problems. Now it appears that some things that were previously logged at level=FINE are now logged at INFO, when they should be DEBUG. I'm working on fixing that...

▼  Doug Cutting added a comment - 03/Jun/06 02:20

With some trepidation, I just committed this.


There were a number of problems with this patch. It changed all level=fine log messages into level=info, rather than level=debug. The build needed to be repaired as well, since logging is performed there, but the standard configuration is not appropriate during build, so I added a build/test log4j configuration. Finally, the changes to bin/hadoop did not name the log files correctly: the correct log file name should be normally set in bin/hadoop-daemon.sh and used in bin/hadoop. I fixed all of these.

In the future, we should not try to make such large changes in the last days before a release. Such changes should be made early in the release cycle. I suspect we will still encounter more problems with this as I now try to make a release and test things with Nutch for back-compatibility, on Windows, etc.


▼  Arun Murthy added a comment - 03/Jun/06 13:24

Apologise for all the troubles... we assumed fine->info, finer->debug, finest->trace mappings; we should have run this through you once.

Next time please throw out the patch if we screw up and let us scramble to fix the mess we created... appreciate your patience. Thanks!

▼  Arun Murthy added a comment - 07/Jun/06 17:52

We seemed to have missed out getMapOutput.jsp in the earlier patch... here's the fix. Thanks!

▼  Sanjay Dahiya added a comment - 23/Jun/06 07:11

New extensions to hadoop logging -

- Rolling based on both time and size.
- compress
- Move rolled over files to DFS


Log4J 1.3 has a better way of defining rollover policies and actions, I have a working implementation of above but they depend on Log4J 1.3. Also the properties file will change to an XML format as the .properties format doesn't support all the configurations yet. Are we willing to move to 1.3 and XML log4j properties ?

▼  Barry Kaplan added a comment - 01/Jul/06 00:16


According to the tomcat 5.5 docs, XML configuration files don't allow you to use naming convention for logs within tomcat. As someone who uses tomcat and hadoop I would prefer not using the xml log.

You can (and should) be more picky about which packages to include in the logging. Tomcat 5.5 uses defines loggers by Engine and Host names. For example, for a default Catalina localhost log, add this to the end of the log4j.properties above. Note that there are

known issues with using this naming convention (with square brackets) in log4j XML based configuration files, so we recommend you use a properties file as described until a future version of log4j allows this convention.

▼  **Sanjay Dahiya** added a comment - 03/Jul/06 19:50

I'm looking at support logging features like (cap on time/size, gzip) and archiving log files into DFS. Log4j 1.3 with XML configurations makes it real easy to implement all these with the RollingPolicies and Triggers separated from appenders. properties file format doesn't allow for specifying RollingPolicies externally for existing Appenders. Are you embedding Tomcat within Hadoop or using Hadoop from a webapp? Is it possible to make tomcat use its own properties file or configure Log4J for the webapp separately in the webapp's class loader?

▼  **Barry Kaplan** added a comment - 06/Jul/06 06:26

I am using Hadoop within tomcat, my guess is there is a way to make hadoop use its own log properties that is separate from tomcat's, but it will be rather annoying to have a separate log4j.properties on a library by library basis.

▼  **Sanjay Dahiya** added a comment - 07/Jul/06 16:40

From what I understand, you are using hadoop client in tomcat. We need a light weight client for embedding in other apps for these use cases. That client can use apps logging configuration. This logging is primarily targeted at Name node, data node, and trackers which generate logs on an ongoing basis. These log configurations need to be separated from Hadoop client in any case.


▼  **Eric Baldeschwieler** added a comment - 07/Jul/06 23:29

Is the client code logging? If so we should file another bug to make this independent of all of the server logging for sure. As barry explains his rig, I think he is actually running the hadoop servers inside tomcat as well. This is a more complicated issue. It may well make sense to support this, don't know enough about the environment to understand the pros and cons. An obvious pro is that he has fewer processes to shepherd. A con is that it isn't something anyone else has worked through or made work.

▼  **Eric Baldeschwieler** added a comment - 07/Jul/06 23:32

Barry, I've filed a new enhancement bug (ability to run datanodes in tomcat) to serve as an umbrella for this issue. I think you've filed another bug on jeti related to this as well. It would be good to tie all the tomcat issues together. I'd be curious to know what others on the list (who know more about java/tomcat) think about this proposal, but I think we should move the discussion off this bug.

<http://issues.apache.org/jira/browse/HADOOP-353> - Run datanode (or other hadoop servers) inside tomcat

▼  **Sanjay Dahiya** added a comment - 25/Jul/06 20:57

A patch for rolling hadoop logs from all nodes to a well defined directory structure in DFS. Log files can be optionally compressed with gzip or zip. Log files are rolled over after a default 10MB or at midnight. The files are rolled over in DFS at a configurable path in a directory structure e.g. -
<archive path>/year/month/day/logfile_index.log.gz

This patch depends on Log4J 1.3, so we may not want to commit it in main trunk yet. It can be used if one doesn't mind using log4j's XML configuration file format. Log4J's DailyRollingFileAppendr will be deprecated in future versions so we will have to use this inevitably.

Please remember to upgrade the log4j.jar file to 1.3 version and remove the old version or this patch will not compile.

▼ People

Assignee:



Sameer Paranjpye

Reporter:



Sameer Paranjpye

Votes:

0 Vote for this issue

Watchers:

3 Start watching this issue

▼ Dates

2/12/23, 9:22 PM

[HADOOP-211] logging improvements for Hadoop - ASF JIRA

Created:

12/May/06 06:16

Updated:

03/Aug/06 17:46

Resolved:

03/Jun/06 02:20