Paper for: Jazoon'07 - Internationale Konferenz für Java-Technologie

jSCSI - A Java iSCSI Initiator

Marc Kramis, Volker Wildi, Bastian Lemke, Sebastian Graf, Halldór Janetzko, Marcel Waldvogel

2007-06-25

Abstract

jSCSI represents an initiator implementation of the iSCSI [1] standard. This short paper describes the current work-in-progress of the jSCSI 1.0 release and gives first benchmarks as well as an outlook for upcoming releases.

1 Introduction

Accessing persistent storage from Java usually means talking to the file system through the frugal Java file system abstraction, i.e., the File class. Currently, there is no platform-independent way to directly talk to a single local or remote storage device, yet alone a device pool. jSCSI aims to fill this gap by implementing the iSCSI protocol right in Java. We believe that firstly, Java is mature enough to cleanly implement well-performing low-level storage protocols and secondly, that it would be very convenient to plug a terabyte-sized iSCSI RAID into the local network and immediately connect to it from any JVM.

The iSCSI protocol defines how a client (iSCSI initiator) accesses a block device on a server (iSCSI target) over a TCP/IP network. It is inspired by the existing SCSI protocol used to access local hard drives or other devices in a block-oriented fashion. Being standardized in April 2004 with RFC 3720 [1], it was quickly adopted, not least because it is believed to offer a better price-performance ratio and fewer infrastructure changes than competing solutions such as fibre channel [2]. Furthermore, recent research indicates that user-level iSCSI initiators can improve performance considerably [3]. The main reason, as argued by the authors, is due to the reduced copy-overhead induced by the user-space to kernel barrier.

jSCSI includes a Java iSCSI initiator, a Java device activity monitoring tool and a preliminary iSCSI backend for the widely used full-text search engine Lucene [4]. Future jSCSI releases shall come with an adaptive storage pool inspired by Sun Microsystem's ZFS [5] as well as a more elaborate jSCSI initiator and iSCSI backend for Lucene. Releasing jSCSI under the Apache License V2.0 [6] to the open source community will allow a bigger audience to work with devices out of Java as they would work with files.

The rest of this extended abstract is organized as follows: Section 2 describes our current work, i.e., jSCSI 1.0. Section 3 introduces the planned features and extensions for jSCSI 2.0. Finally, Section 4 summarizes our preliminary findings and concludes this extended abstract.

2 Current Work

The first release of jSCSI provides a simple interface for a device, i.e., Device, as listed in Listing 1. A Device implementation must comply with the following semantics: Multiple threads can concurrently call the *read(...)*, *write(...)*, and *getX()* methods. Each method call of one thread is executed synchronously. Operation queueing and reordering is the task of the device implementation whereas caching is the responsibility of the upper layers.

Listing 1: Java Device Interface

```
public interface Device {
  public void open();
  public String getName();
  public int getBlockSize();
  public long getBlockCount();
  public void read(final long address, final byte[] buffer);
  public void write(final long address, final byte[] buffer);
  public void close();
}
```

The jSCSI 1.0 initiator implements the Device interface and binds each device to one iSCSI target. The jSCSI 1.0 storage pool is a small extension to map a device to a striped or mirrored RAID currently consisting of two or four devices for improved performance or reliability. The initiator can be configured to establish multiple sessions to various targets. Each session uses exactly one TCP connection and operates synchronously. The initiator supports the login operational negotiation as well as the full feature phase to configure the behavior of the session and to transmit data. A set of parsers and serializers together with a state machine per session and connection assure the proper execution of the iSCSI protocol.

Besides the basic functionality for accessing an iSCSI device from Java, jSCSI 1.0 also comes with a device monitoring tool that allows to remotely visualize the activity of multiple jSCSI initiators. The jSCSI 1.0 initiator therefore sends a dump of the Device interface method calls (excluding the buffer contents) to a monitoring instance over a TCP connection. The monitoring instance running in Eclipse [7] then interactively displays the read and write touches on the device in an Eclipse view.

Finally, we implemented a device-based backend for Lucene, i.e., a DeviceDirectory. This allows to store a Lucene full-text index directly on a raw iSCSI target. The Lucene example demonstrates how jSCSI can be used, what performance it achieves, and how complex the setup and maintenance is compared to a filesystem backend. The current DeviceDirectory works according to the log-structured, i.e., copy-on-write principle only appending new blocks at the end.

Preliminary benchmarking results for the device and filesystem backend of Lucene as well as a few read and write operations on a single or multiple targets with both the Java and Open-iSCSI initiator [8] are listed in Table 1. The numbers give a first impression of the performance available with an average computer and networking equipment. Final results with a detailed analysis of all use cases are left to future work. Note that jSCSI uses a simple LRU cache but no prefetching and that cache hits are only available for searching a Lucene index.

Description	unit	min	max	avg	stddev	conf95	runs
Read 40 kB							
jSCSI	ms	2	14	3,34	1,32	[3,08;3,60]	100
Open-iSCSI	ms	4	15	5,29	1,32	[5,03;5,55]	100
Read 400 kB							
jSCSI	ms	22	24	22,48	0,52	[22,38;22,58]	100
Open-iSCSI	ms	5	16	6,79	3,19	[6,17;7,41]	100
jSCSI write 8MB							
No RAID, 1 target	ms	3889	6987	4347,55	398,42	[4322,86;4372,24]	1000
RAID 0, 2 targets	ms	2346	4947	2803,46	330,12	[2783,00;2823,92]	1000
RAID 1, 2 targets	ms	4663	9447	5313,72	487,65	[5283,50;5343,95]	1000
Lucene *Getting started*							
Build file system index	ms	523	1270	676,40	153,80	[609,00;743,80]	20
Build jSCSI index	ms	6363	7889	6927,95	468,25	[6722,73;7133,17]	20
Search file system index	ms	2	68	7,05	14,09	[0,87;13,23]	20
Search jSCSI index	ms	5	10	9,45	1,36	[8,85;10,05]	20

3 Future Work

Several improvements are planned for jSCSI 2.0. The Device interface currently comes with a contract for synchronous interaction on a per-thread basis. Asynchronous, i.e., non-blocking I/O semantics will allow applications to scale better due to a more efficient use of available CPU resources. There will also be an abstract Device implementation coming along with queuing and prefetching support and other common features for all available devices.

The jSCSI 2.0 initiator will support multiple pending operations per connection. This works according to the pipelining principle that allows to substantially increase the throughput. Another upcoming feature is the security negotiation phase and user authentication. Timers and keep-alive pings will assure that the connection is not torn down unexpectedly. Multiple TCP connections per session will be implemented when the common iSCSI targets (such as the Enterprise iSCSI Target [9]) will start supporting this feature. Note that multiple connections can improve the resilience of a session as well as the throughput because of the multipathing effect [10]. Finally, we continue working on optimization toward smaller memory footprint as well as reducing garbage collection overhead and CPU consumption.

The jSCSI 2.0 storage pool is planned to be extended to a full-fledged storage pool similar to ZFS's storage pool. The main reason for duplicating the pool functionality in Java is the ease of rapid prototyping coming along with Java, especially compared to in-kernel development. One of the next features to develop is the ability to balance writes across multiple devices according to the device space usage, activity, and latency statistics.

References

[1] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. Internet Small Computer Systems Interface (iSCSI). RFC 3720 (Proposed Standard), April 2004. Updated by RFC 3980.

[2] Adaptec. The Cost Benefits of an iSCSI SAN, 2003.

[3] R. Sohan and S. Hand. A User-Level Approach To Network Attached Storage. In *In Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary*, pages 108–114, 2005.

[4] Lucene. At http://lucene.apache.org.

[5] ZFS. http://www.sun.com/2004-0914/feature.

[6] Apache License V2.0. http://www.apache.org/licenses/LICENSE-2.0.

[7] Eclipse. http://www.eclipse.org.

[8] Open-iSCSI. At http://www.open-iscsi.org/.

[9] The iSCSI Enterprise Target Project. At http://iscsitarget.sourceforge.net.

[10] A. Dailey and S. Tracy. Using ISCSI Multipathing in the Solaris 10 Operating System. Technical report, SUN BluePrints TM OnLine. December 2005, 2005.