Performance Benchmark and Scalability

James Min

Senior Consultant / Sales Engineer, Liferay, Inc.

LIFERAY

LIFERAY WEST COAST SYMPOSIUM

Goals: A Proper Benchmark

- Real world benchmarks
  - Content intensive scenarios
  - Social network and collaboration scenarios

- Benchmarked reference architecture
  - Small, medium, and large deployments
  - HA deployments

- Dispel Misinformation
  - “Does Liferay Portal scale beyond a small intranet?”
  - “Can Liferay handle more than 25 concurrent users?”

A Production-like Environment

Hardware

1 - Web Server

1 Intel Xeon Quad-core 2.4Ghz, 2GB

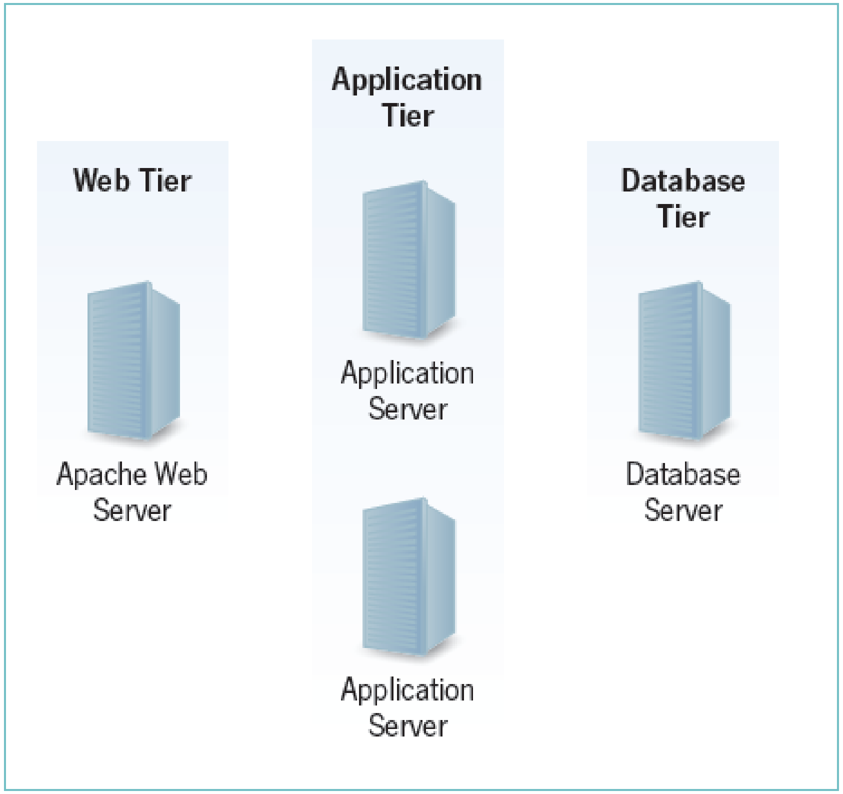
2 - Application Server

2 Intel Xeon Quad-core 2.4Ghz, 8GB

1 - Database Server

2 Intel Xeon Quad-core 2.4Ghz, 4GB, 15k RPM 250GB HD

Tomcat 6.0.18, JDK 1.6.0_12, and MySQL 5.0.67



Realistic Scenarios

- Transactional Performance
  - Login throughput
  - Concurrent users

- Content management scenarios
  - 70% anonymous access, 30% authenticated access
  - Liferay Web Content
    - 85% browsing, 15% content creation/update
    - 10,000 articles

Realistic Scenarios

- Collaboration scenarios
  - 40% anonymous access, 60% authenticated access
  - Liferay Blogs
    - 75% views, 15% comments, 10% creation
    - 100,000 blog entries
  - Liferay Wiki
    - 75% views, 15% comments, 10% creation
    - 100,000 wiki entries
  - Liferay Message Boards
    - 70% views, 30% reply/new posts
    - 500,000 posts across 100 categories

Testing Tools

- Data Set Generator
  - Configurable generation of test data
    - 1MM users generated randomly from US census data (max 400MM users)
    - 1MM message board posts
    - 100k blog posts
    - 100k wiki articles
    - 1000 web content

- Load Generator (Grinder)
  - Python load scripts enforcing ramp up time, think time, etc
  - Able to scale to clusters of test clients/injectors
  - Must use Linux to inject load
  - Validates successful tests via checksums on the response.

1,000,000 Unique Users

- Liferay – Authenticated Access
  - Scenario:
    - User arrives on Liferay.com homepage.
    - User authenticates with portal.
    - Portal forwards authenticated user to homepage.
  - 1MM unique users in database
  - Throughput: 38.4 login/sec; 138,240 logins in 1 hour
  - Average time: 938ms/execution
  - Average CPU load: 64%

Methodology

Liferay utilized The Grinder load testing tool and its distributed load injectors. In all test scenarios, the injectors ramped up users at a rate of one user every 100 milliseconds until the maximum concurrent user load has been achieved.

The benchmark data was gathered after an initial ramp up time of 5 minutes to initialize all application elements and warm up all injectors. As part of data gathering, the following statistics were gathered:

- CPU statistics on web, application, and database servers.
- JVM garbage collection information via Visual VM and garbage collector logs.
- Average transaction times, standard deviations, and throughput from The Grinder console.

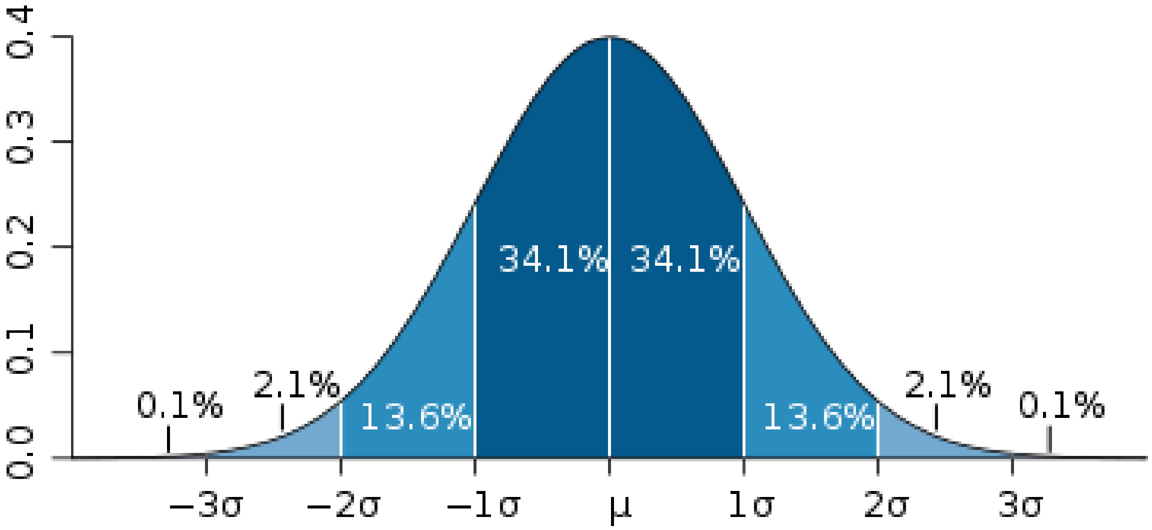
Methodology

- Although the benchmark environment consisted of two application servers, a single application server was used to perform most tests.

- Once the tests determine the max performance of a single server, Liferay utilized the second application server to prove the linear scalability hypothesis.

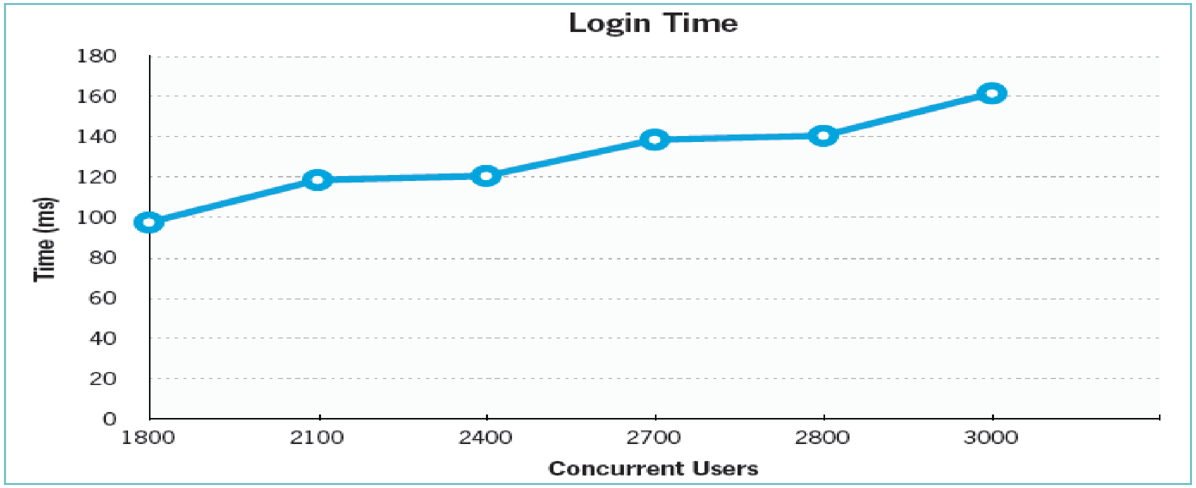
- In theory, doubling the available hardware will double the maximum concurrent supportable user load.

Standard Deviation



Login Transaction

CONCURRENT USERS	DURATION (MIN)	μ(ms)	σ(ms)	2σ (ms)	THROUGHPUT (TPS)	CPU UTILIZATION (%)
1,800	30	97.9	44.2	186.3	51	36
2,100	30	119	199	517	52.6	44
2,400	30	121	197	515	59.6	50
2,700	30	139	199	537	67.4	58
2,800	30	141	177	495	70	63
3,000	30	162	206	574	74.2	71
3,200	30	310	367	1044	76	78
3,300	30	430	452	1334	79.1	81



Login Test Results

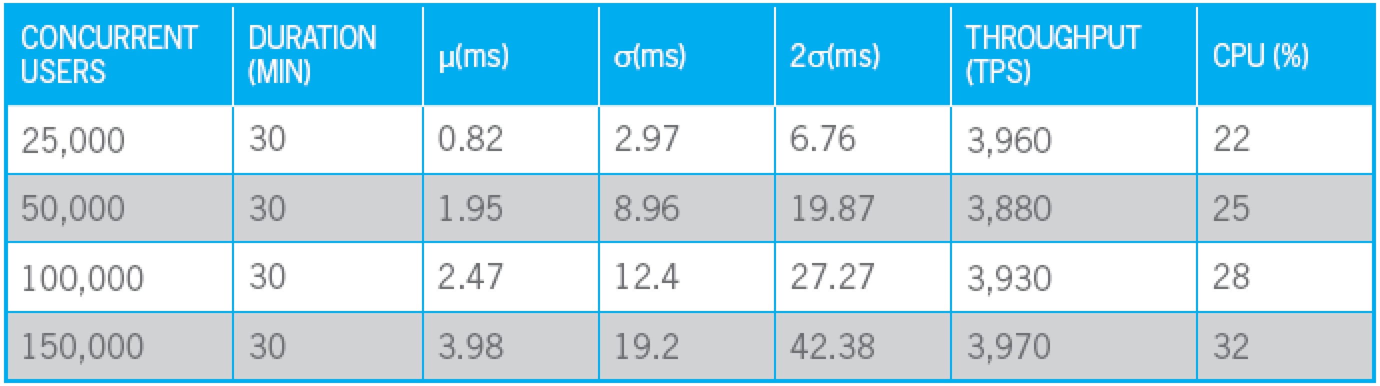
- The mean time for login remains less than 200ms as we approach the performance inflection point.

- At 3000 concurrent users, we have a mean time (μ) of 162ms and 95% of the logins (2σ) below 600ms.

- The optimal performance point for this test scenario occurs somewhere between 3000 to 3200 concurrent users.

- At 3200 users, we see 2σ increasing to over 1s, moving above the acceptable threshold. At this inflection point, we see CPU utilization at roughly 71% on the application server.

Web Content Management



CONCURRENT USERS	DURATION (MIN)	μ(ms)	σ(ms)	2σ(ms)	THROUGHPUT (TPS)	CPU (%)
25,000	30	0.82	2.97	6.76	3,960	22
50,000	30	1.95	8.96	19.87	3,880	25
100,000	30	2.47	12.4	27.27	3,930	28
150,000	30	3.98	19.2	42.38	3,970	32

TABLE 3 – BROWSING FOR CONTENT ANONYMOUSLY

Message Boards

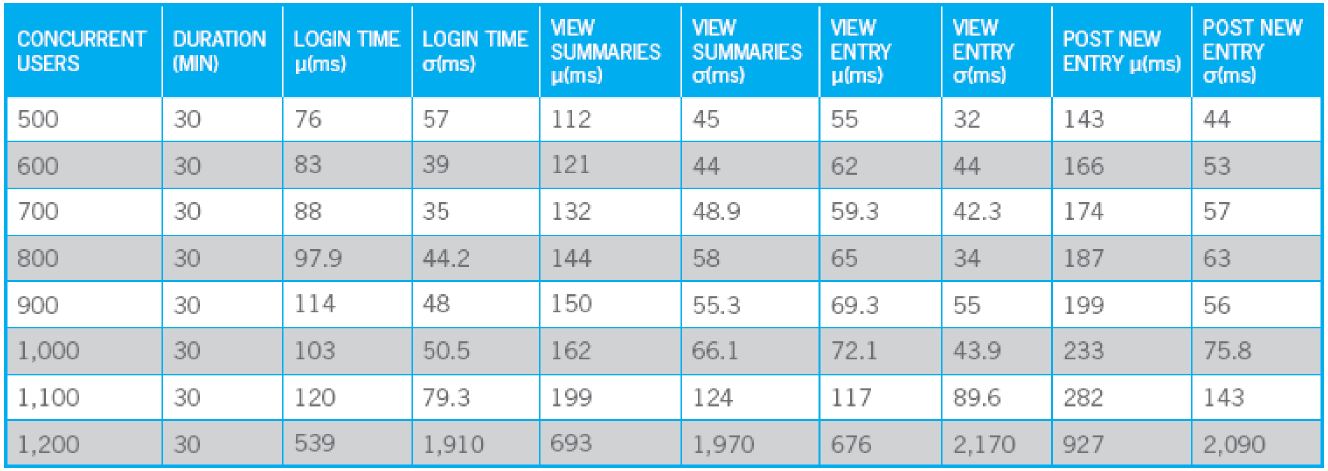
CONCURRENT USERS	DURATION (MIN)	LOGIN TIME µ(ms)	LOGIN TIME σ(ms)	BROWSE CATEGORY µ(ms)	BROWSE CATEGORY σ(ms)	BROWSE THREAD µ(ms)	BROWSE THREAD σ(ms)	BROWSE POSTS µ(ms)	BROWSE POSTS σ(ms)
500	30	96.6	59.3	117	47.3	188	65.5	291	96.8
600	30	104	154	122	155	201	233	294	93.5
700	30	102	104	123	121	202	180	307	111
800	30	109	107	136	113	229	164	351	217
900	30	117	80.9	149	95.7	264	166	390	223
1,000	30	143	130	192	137	340	253	526	370
1,100	30	139	181	180	156	332	252	511	358
1,200	30	655	193	202	235	490	341	774	507
1,300	30	889	1,570	1,180	3,210	2,120	4,150	3,050	4,980

TABLE 5 – MESSAGE BOARDS PART 1

CONCURRENT USERS	POST THREAD µ(ms)	POST THREAD σ(ms)	REPLY THREAD µ(ms)	REPLY THREAD σ(ms)	TOTAL µ(ms)	TOTAL σ(ms)	TOTAL 2σ(ms)	CPU (%)
500	81	42.1	328	124	1,101.6	435	1,971.6	24
600	83.6	43.1	334	90.5	1,138.6	769.1	2,676.8	28
700	86	38.3	342	118	1,162	672.3	2,506.6	31
800	110	135	392	242	1,327	978	3,283	38
900	121	100	441	249	1,482	914.6	3,311.2	48
1,000	168	155	621	443	1,990	1,488	4,966	56
1,100	174	260	593	472	1,929	1,679	5,287	63
1,200	254	243	866	574	3,241	2,093	7,427	73
1,300	3,450	6,710	4,210	5,740	14,899	26,360	67,619	82

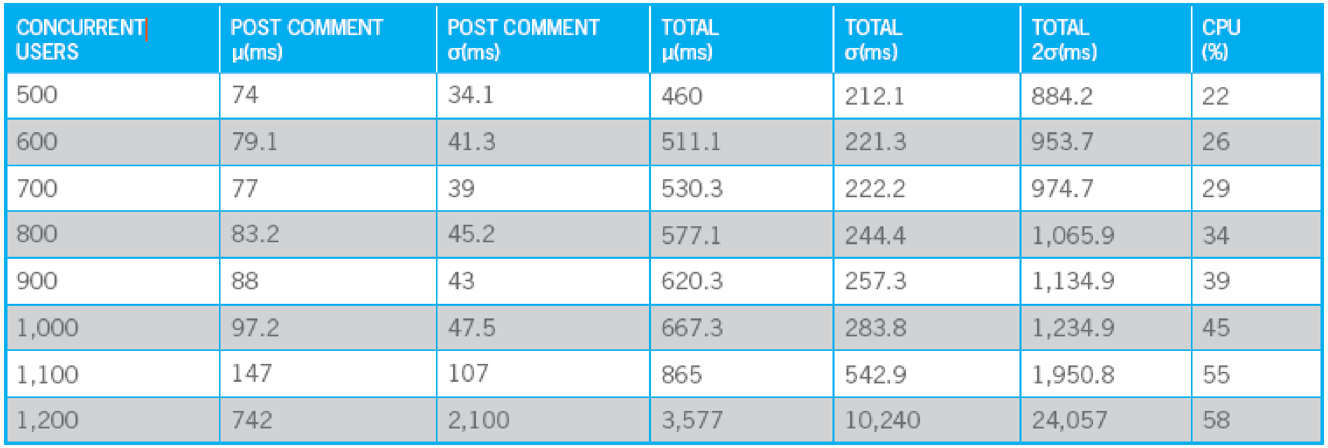
TABLE 6 – MESSAGE BOARDS PART 2

Blogging



CONCURRENT USERS	DURATION (MIN)	LOGIN TIME µ(ms)	LOGIN TIME σ(ms)	VIEW SUMMARIES µ(ms)	VIEW SUMMARIES σ(ms)	VIEW ENTRY µ(ms)	VIEW ENTRY σ(ms)	POST NEW ENTRY µ(ms)	POST NEW ENTRY σ(ms)
500	30	76	57	112	45	55	32	143	44
600	30	83	39	121	44	62	44	166	53
700	30	88	35	132	48.9	59.3	42.3	174	57
800	30	97.9	44.2	144	58	65	34	187	63
900	30	114	48	150	55.3	69.3	55	199	56
1,000	30	103	50.5	162	66.1	72.1	43.9	233	75.8
1,100	30	120	79.3	199	124	117	89.6	282	143
1,200	30	539	1,910	693	1,970	676	2,170	927	2,090

TABLE 7 – BLOGS PART 1



CONCURRENT USERS	POST COMMENT µ(ms)	POST COMMENT σ(ms)	TOTAL µ(ms)	TOTAL σ(ms)	TOTAL 2σ(ms)	CPU (%)
500	74	34.1	460	212.1	884.2	22
600	79.1	41.3	511.1	221.3	953.7	26
700	77	39	530.3	222.2	974.7	29
800	83.2	45.2	577.1	244.4	1,065.9	34
900	88	43	620.3	257.3	1,134.9	39
1,000	97.2	47.5	667.3	283.8	1,234.9	45
1,100	147	107	865	542.9	1,950.8	55
1,200	742	2,100	3,577	10,240	24,057	58

TABLE 8 – BLOGS PART 2

HA: The Different Degrees

Types of Standby

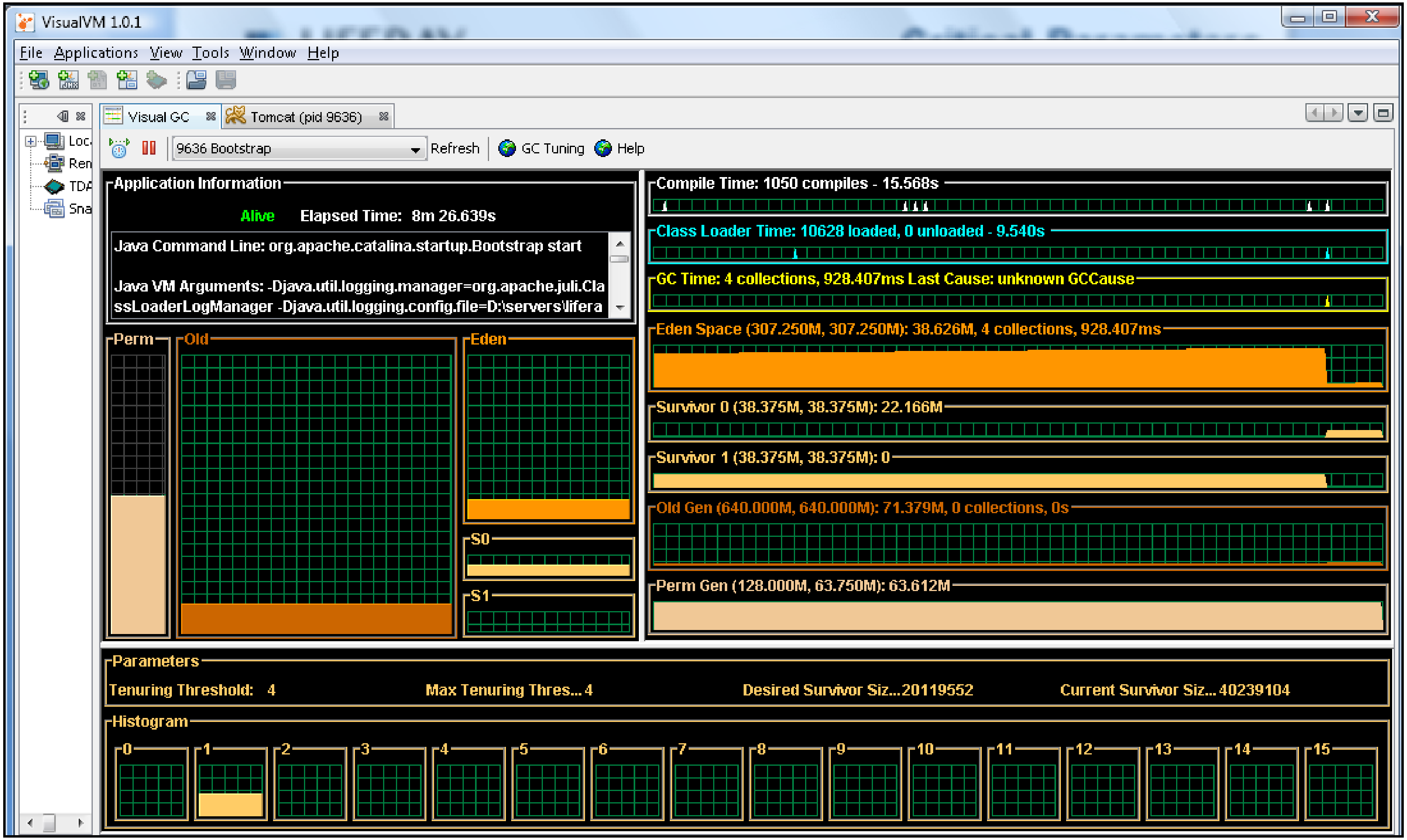
- Hot-Hot: immediate availability
- Hot-Warm: available after DNS propagation
- Hot-Cold: i.e. 3-4 hours to come up

Tuning Best Practices

Need:

- Load generator (The Grinder, JMeter, LoadRunner, etc.)
- Java Profiler (VisualVM, JProfiler, YourKit, etc.)

VisualVM



Application Information

Alive Elapsed Time: 8m 26.639s

Java Command Line: org.apache.catalina.startup.Bootstrap start

Java VM Arguments: -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.util.logging.config.file=D:\servers\lifera

Compile Time: 1050 compiles - 15.568s

Class Loader Time: 10628 loaded, 0 unloaded - 9.540s

GC Time: 4 collections, 928.407ms Last Cause: unknown GCCause

Eden Space (307.250M, 307.250M): 38.626M, 4 collections, 928.407ms

Survivor 0 (38.375M, 38.375M): 22.166M

Survivor 1 (38.375M, 38.375M): 0

Old Gen (640.000M, 640.000M): 71.379M, 0 collections, 0s

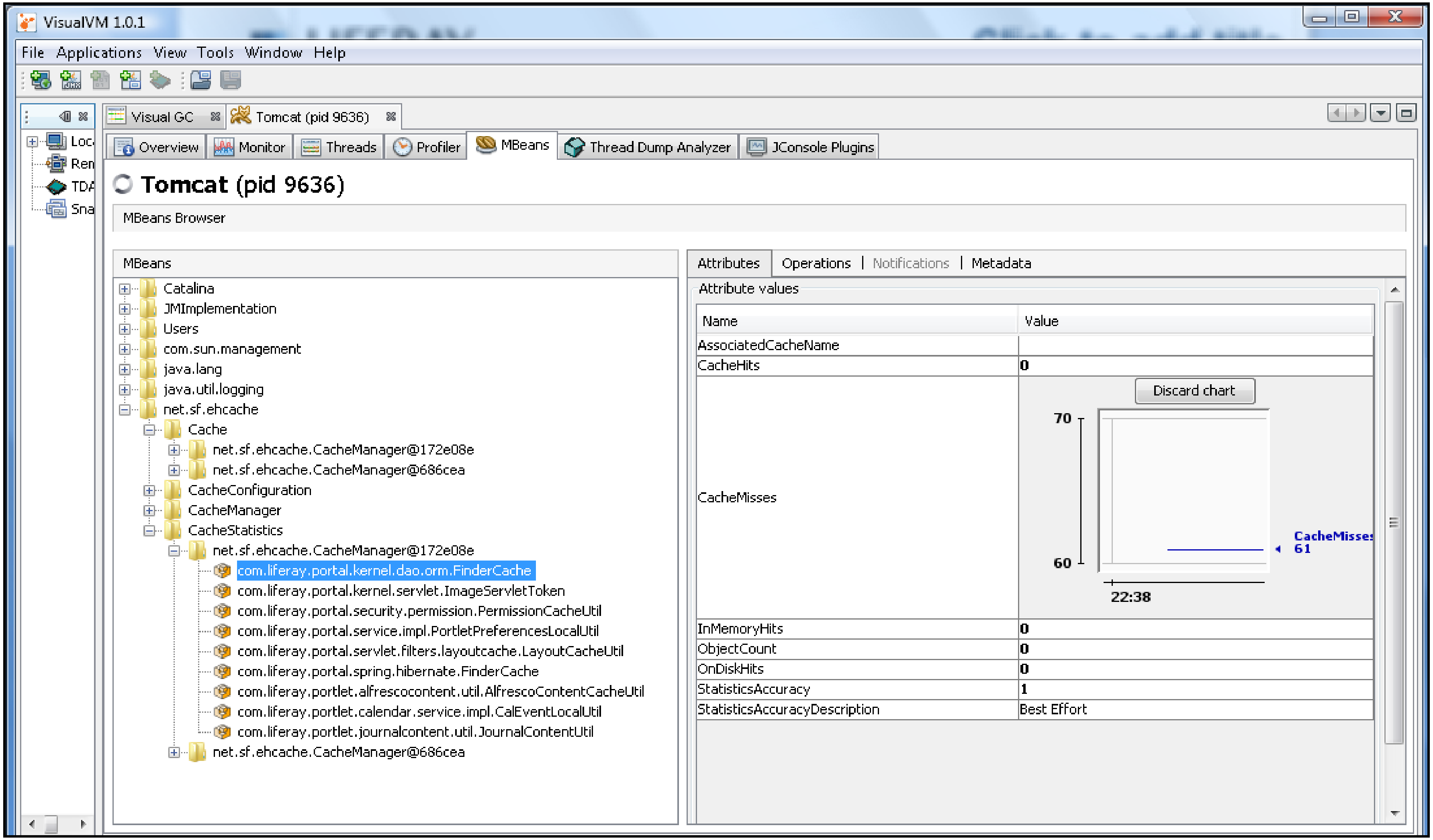
Perm Gen (128.000M, 63.750M): 63.612M

Parameters

Tenuring Threshold: 4 Max Tenuring Thres... 4 Desired Survivor Siz...20119552 Current Survivor Siz... 40239104

Histogram

VisualVM



Tomcat (pid 9636)

MBeans Browser

MBeans

```
⊞ Catalina
⊞ JMImplementation
⊞ Users
⊞ com.sun.management
⊞ java.lang
⊞ java.util.logging
⊟ net.sf.ehcache
    ⊟ Cache
        ⊞ net.sf.ehcache.CacheManager@172e08e
        ⊞ net.sf.ehcache.CacheManager@686cea
    ⊞ CacheConfiguration
    ⊞ CacheManager
    ⊟ CacheStatistics
        ⊟ net.sf.ehcache.CacheManager@172e08e
            com.liferay.portal.kernel.dao.orm.FinderCache
            com.liferay.portal.kernel.servlet.ImageServletToken
            com.liferay.portal.security.permission.PermissionCacheUtil
            com.liferay.portal.service.impl.PortletPreferencesLocalUtil
            com.liferay.portal.servlet.filters.layoutcache.LayoutCacheUtil
            com.liferay.portal.spring.hibernate.FinderCache
            com.liferay.portlet.alfrescocontent.util.AlfrescoContentCacheUtil
            com.liferay.portlet.calendar.service.impl.CalEventLocalUtil
            com.liferay.portlet.journalcontent.util.JournalContentUtil
        ⊞ net.sf.ehcache.CacheManager@686cea
```

Attribute values

Name	Value
AssociatedCacheName	
CacheHits	0
CacheMisses	
InMemoryHits	0
ObjectCount	0
OnDiskHits	0
StatisticsAccuracy	1
StatisticsAccuracyDescription	Best Effort

Capacity: Best Practices

- Create test scenarios that reflect actual usage
- Load test and identify bottlenecks
- Address the bottlenecks
- Rinse, repeat.

There are no magic numbers

Disaster Recovery

Make sure you have a Disaster Recovery plan and test it every quarter or several weeks!

Will Liferay Scale?

Yes! If you:

- Develop valid test scenarios
- Identify and address bottlenecks
- Procure the appropriate hardware and resources

Caching Mechanisms

- Local memory cache with disk overflows
- Replicated distributed cache
- Sharded distributed cache

Local Caching Frameworks

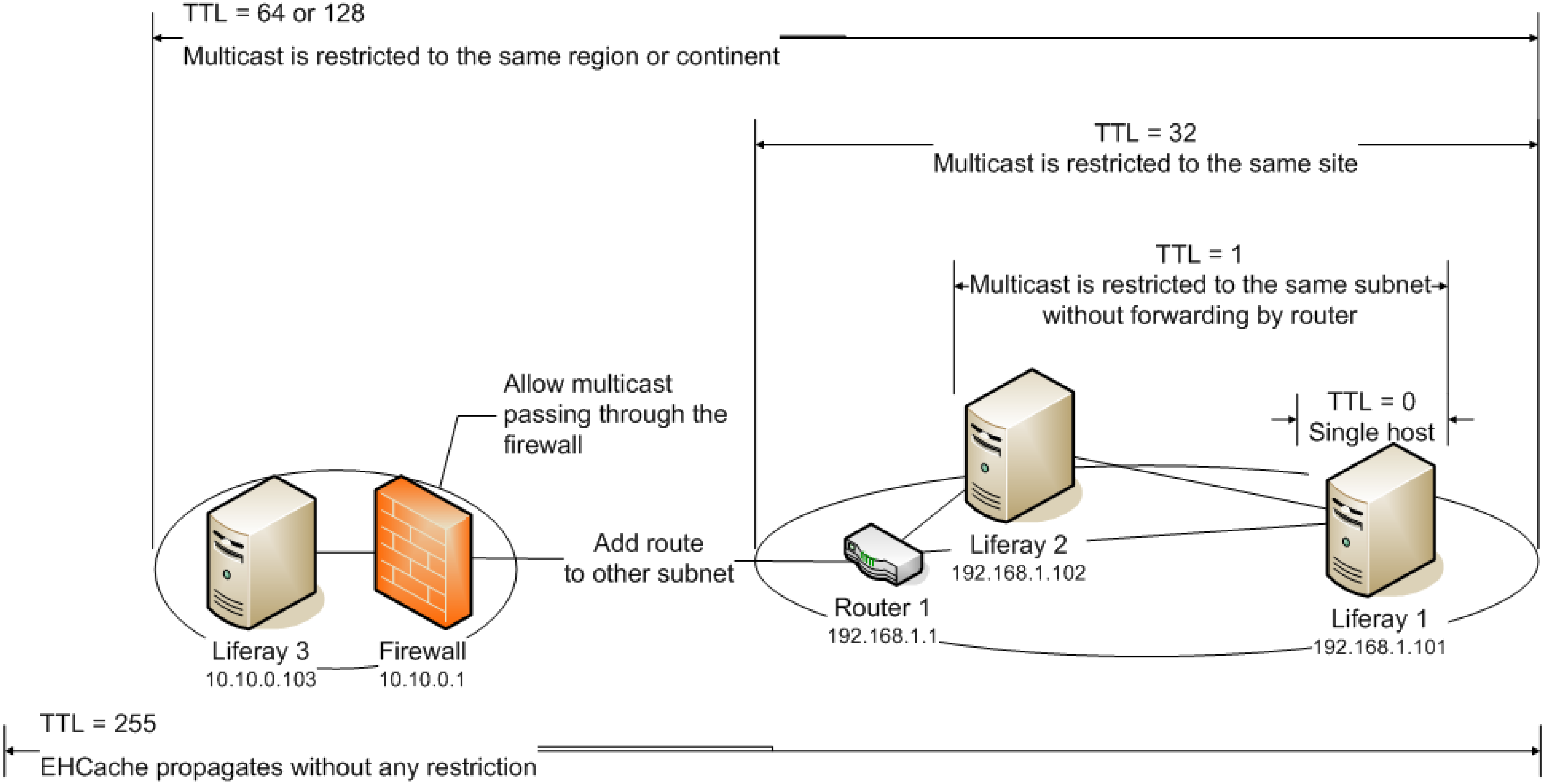
- Developers write what to cache, configuration drives how to cache.
  - Ehcache

```
CacheManager cacheMgr = new CacheManager(configurationURL);
Cache cache = cacheMgr.getEhcache(cacheName);
Object value = cache.get(objectKey);
```

Replicated Distributed Cache

- **Default Ehcache Replication**
  - Easy to configure
  - Multicast discovery with RMI Replication
  - Scaling considerations due to multicast mechanism overhead
  - 1 replication thread per cached object

Liferay + Replicated Ehcache



Sharded Distributed Cache

- **Terracotta**
  - Highly scalable, commercial open source solution.
  - Supports both sharded and replicated modes
  - Rich set of monitoring tools to manage cache performance.
  - Partitioned cache: 1 cache per entity

- **Memcached**
  - Popular open source solution used by Facebook, Google, and other large deployments
  - Max 2MB cached object size
  - Use multiple languages to access cache
  - “Roll your own” tools and strategies
  - Cache is 1 large cache, no partitioning

Challenges of Multi-Site HA

WAN’s, replication, and distribution…

Things to consider and do your homework on:

- Accomplishing multi-site HA with database replication, WAN clustering
- Challenges of database replication
- Challenges of WAN cache replication
    - Make sure cache replicates properly
    - DB replicate properly
    - Clustering in app server, session replication

Online Resources

Liferay Portal Performance Whitepaper:

http://www.liferay.com/documentation/additional-resources/whitepapers

How Do I Cluster Liferay With Terracotta?:

http://www.liferay.com/web/mika.koivisto/blog/-/blogs/how-do-i-cluster-liferay-with-terracotta

Questions?