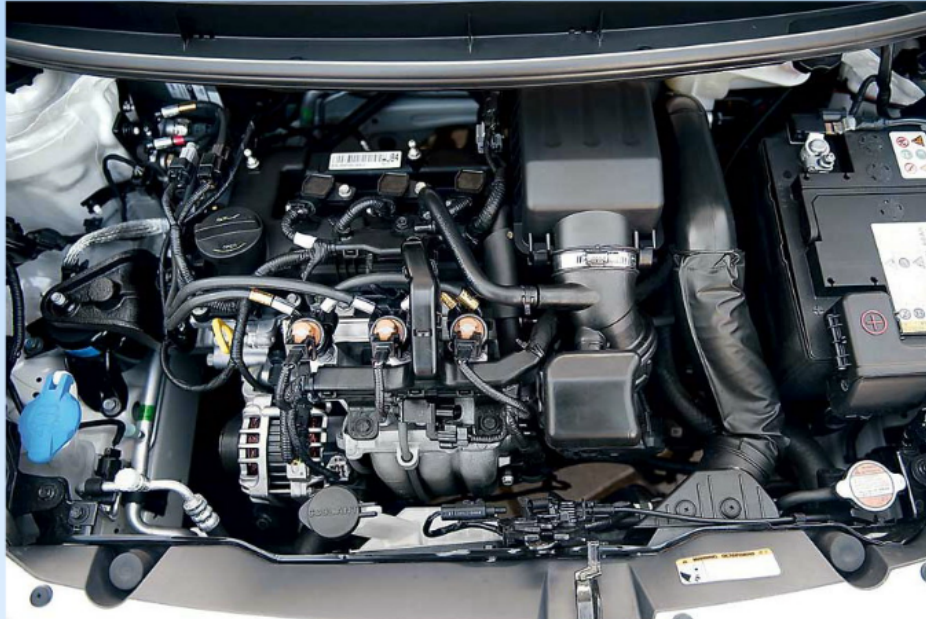






# JVM is complex!



Java HotSpot, Compressed Oops, Branch Prediction, Null Check Elimination, Loop Unrolling, Inlining Methods, Thread fields and Thread Local Storage, Uncontended Synchronization, and so on.

FileViewMemoryCPUSettingsToolsHelp

MainWorkloadSimulator-2017-05-02.snapshot [/Users/Isliwko/Snapshots] - YourKit Java Profiler 12.0.2

CPUThreadsMemoryGarbage CollectionExceptionsProbesInspectionsSummary

Class or method name (?): com.masb

Name	Time (ms)	Avg. Time (ms)	Own Time (ms)	Invocation Count		
com.masb.utils.ObjectUtils\$.selectRandomObjectByWeightInner(Map, double)	93,657	5 %	8	30,125	2 %	10,515
com.masb.workload.simulator.actor.helper.AgentUtils\$.createRecommendation\$1(Task, Node, Res	165,562	8 %	0	14,522	1 %	1,767,922
com.masb.workload.simulator.function.NodeScoreFunctions\$.anonfun\$getScoreFuntion\$6\$adapt	51,458	3 %	0	11,380	1 %	1,839,117
com.masb.domain.TaskConstraints.checkConstraints(NodeAttributes)	87,738	5 %	0	10,580	1 %	1,769,765
com.masb.domain.ResourcesVector.add(Iterable)	13,230	1 %	0	10,010	1 %	4,231,285
com.masb.workload.simulator.actor.helper.AgentUtils\$\$\$Lambda\$389.apply(Object)	22,437	1 %	0	9,579	0 %	1,767,922
com.masb.domain.CandidateNodeRecommendation.hashCode()	14,847	1 %	0	8,861	0 %	982,905
com.masb.workload.simulator.function.NodeScoreFunctions\$.C2(Task, Node, ResourcesVector)	16,749	1 %	0	8,460	0 %	1,839,117
com.masb.domain.ResourcesVector.<init>(double[])	8,006	0 %	0	8,006	0 %	6,909,691
com.masb.domain.ResourcesVector.\$plus(ResourcesVector)	30,890	2 %	0	7,200	0 %	4,231,285
com.masb.workload.simulator.context.ContextData.getTask(String)	58,112	3 %	0	6,973	0 %	4,948,801
com.masb.domain.ResourcesVector.overflows(ResourcesVector)	6,532	0 %	0	6,532	0 %	3,362,768
com.masb.domain.ResourcesVector.isOverflowedBy(Iterable)	10,525	1 %	0	6,380	0 %	253,571
com.masb.domain.ResourcesVector.add(Seq)	19,115	1 %	0	5,885	0 %	4,231,285
com.masb.domain.NodeAttributes.getOrElse(String, String)	29,136	1 %	0	5,490	0 %	624,893

Back TracesCallees ListMerged Callees

Back traces for method selected in the upper table

Name	Time (ms)	Invocation Count		
com.masb.utils.ObjectUtils\$.selectRandomObjectByWeightInner(Map, double)	93,657	100 %	10,515	100 %
com.masb.utils.ObjectUtils\$.selectRandomUniqueObjectsByWeightInner\$1(Seq, Map, int, double)				
com.masb.utils.ObjectUtils\$.selectRandomUniqueObjectsByWeight(Map, int)				
com.masb.workload.simulator.actor.helper.AgentUtils\$.selectCandidateNodesForTaskMigration(Task, Function3, Integer, Iterable)				
com.masb.workload.simulator.actor.BrokerAgentActor\$.selectCandidateNodesForTaskMigration(Task, Integer, ContextData)				
com.masb.workload.simulator.actor.base.BaseNodeAgentActor.getCandidateNodeRecommendationsForTaskMigration(Task)				
com.masb.workload.simulator.actor.NodeAgentActor.processAllTick(NodeAgentAllTick)				
com.masb.workload.simulator.actor.base.BaseNodeAgentActor\$\$\$anonfun\$receive\$1.applyOrElse(Object, Function1)				
com.masb.workload.simulator.actor.base.BaseNodeAgentActor.aroundReceive(PartialFunction, Object)				
akka.actor.ActorCell.receiveMessage(Object)				
akka.actor.ActorCell.invoke(Envelope)				
akka.dispatch.Mailbox.processMailbox(int, long)				

Legend

Non-filtered method

Filtered method

Excluded by adaptive tracing

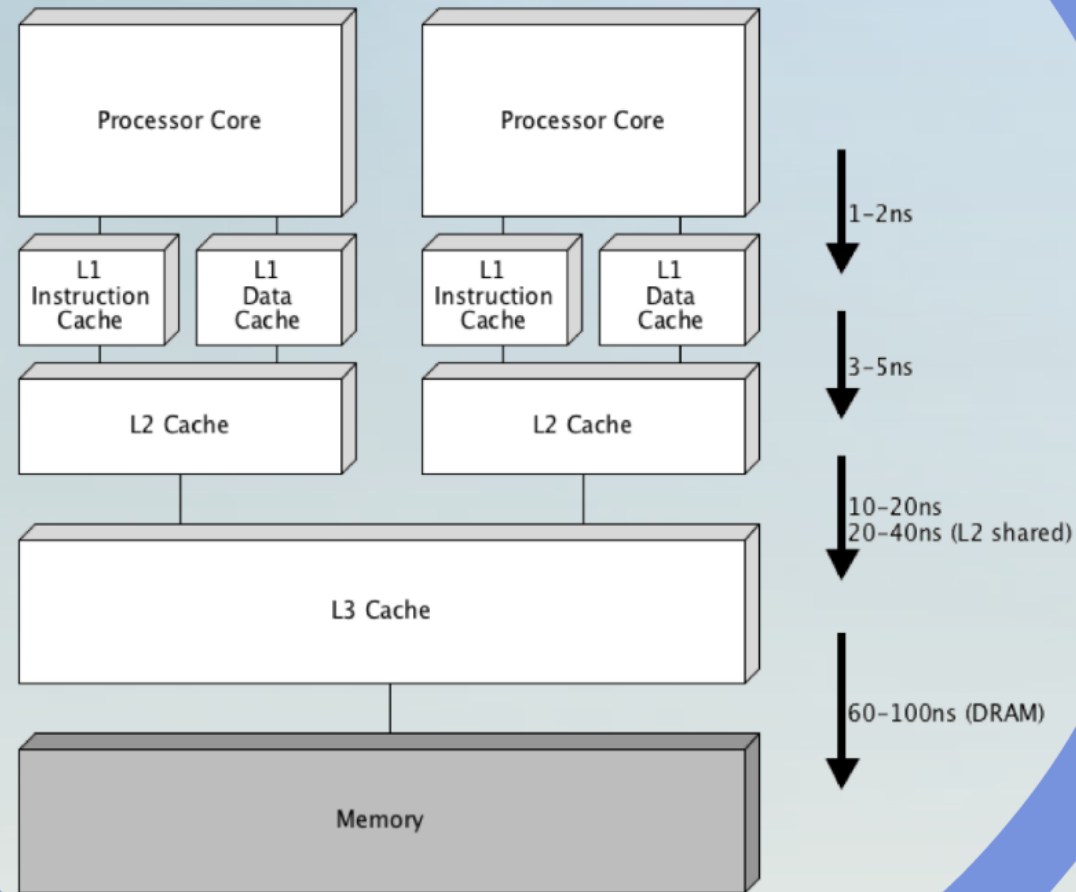
Filters...

Method listMethod back traces: find out where a particular method was called

# Profiler is your friend!

# Enter the Matrix

(Context switch vs CPU caches)







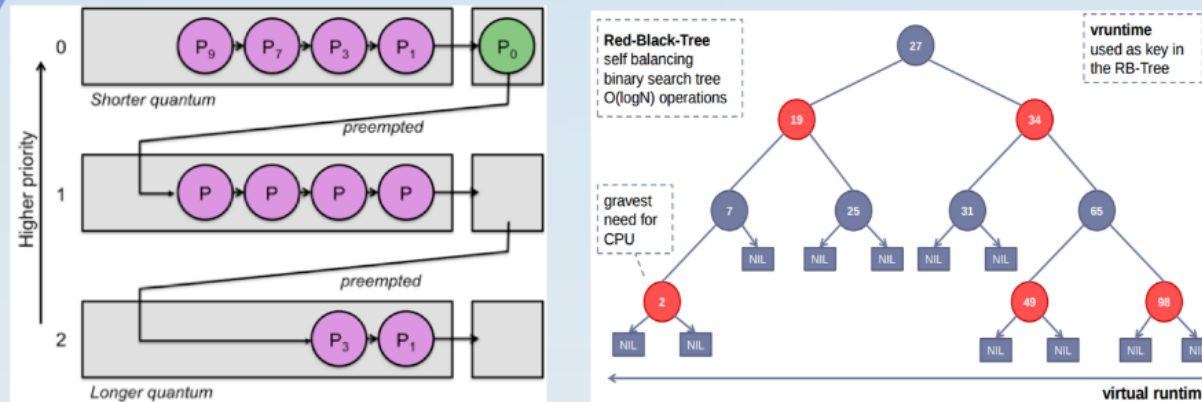
w1518355@compute20:~



```
[w1518355@compute20 ~]$ pidstat -tw -p 119835 5 1 | grep Average | (read -r; printf "%s\n" "$REPLY"; sort -nr -k 5)
```

Average:	UID	TGID	TID	cschw/s	nvcschw/s	Command
Average:	115183558	-	54923	366.27	14.77	__java
Average:	115183558	-	54984	318.16	5.59	__java
Average:	115183558	-	54917	239.52	1.20	__java
Average:	115183558	-	54946	239.12	1.40	__java
Average:	115183558	-	55005	234.13	5.39	__java
Average:	115183558	-	54979	228.94	9.78	__java
Average:	115183558	-	54978	224.15	3.39	__java
Average:	115183558	-	114505	210.98	2.00	__java
Average:	115183558	-	54955	196.41	1.00	__java
Average:	115183558	-	74571	188.82	3.39	__java
Average:	115183558	-	74574	188.02	3.19	__java
Average:	115183558	-	61738	186.03	27.54	__java
Average:	115183558	-	54913	179.24	1.60	__java
Average:	115183558	-	79612	162.67	1.00	__java
Average:	115183558	-	75930	162.67	0.60	__java
Average:	115183558	-	54994	160.28	7.39	__java
Average:	115183558	-	54926	155.69	2.59	__java
Average:	115183558	-	55014	148.10	17.17	__java
Average:	115183558	-	54991	147.31	9.78	__java
Average:	115183558	-	54951	143.11	15.77	__java
Average:	115183558	-	73850	138.92	2.40	__java
Average:	115183558	-	54986	134.33	5.39	__java
Average:	115183558	-	74581	129.94	7.58	__java
Average:	115183558	-	103672	129.94	2.40	__java
Average:	115183558	-	74576	118.96	1.80	__java
Average:	115183558	-	55022	117.17	14.97	__java
Average:	115183558	-	61743	116.97	63.67	__java
Average:	115183558	-	54970	115.57	12.77	__java
Average:	115183558	-	74582	114.57	30.34	__java
Average:	115183558	-	54988	112.57	2.59	__java
Average:	115183558	-	55000	111.18	2.00	__java
Average:	115183558	-	55023	107.19	21.76	__java
Average:	115183558	-	82353	100.20	4.59	__java
Average:	115183558	-	119886	97.80	0.00	__java

# OS Scheduler matters too! (a bit)



Sources: <https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>,  
<https://stackoverflow.com/questions/34442691/linux-kernel-task-h-load>

*“Let's face it - the current scheduler has the same old basic structure that it did almost 10 years ago, and yes, it's not optimal, but there really aren't that many real-world loads where people really care. I'm sorry, but it's true.” Linus. (<http://tech-insider.org/linux/research/2001/1215.html>)*

(742 commits to fair.c since November 2011)

## ***The Good, the Bad...***

```
val numbers = 1 to 10000
```

```
numbers.par.foreach{calculate(_)}
```

```
val futures = for (i <- numbers) yield Future {calculate(i)}  
Await.ready(Future.sequence(futures), Duration.Inf)
```



## *...and the Ugly (Akka Streams)*

```
Source(numbers)
  .groupBy(cpuCount, number => number.hashCode % cpuCount)
  .async ← Beware of auto-folding
  .via(Flow[Int].map(calculate(_)))
  .async
  .mergeSubstreams
  .runWith(Sink.ignore)
```

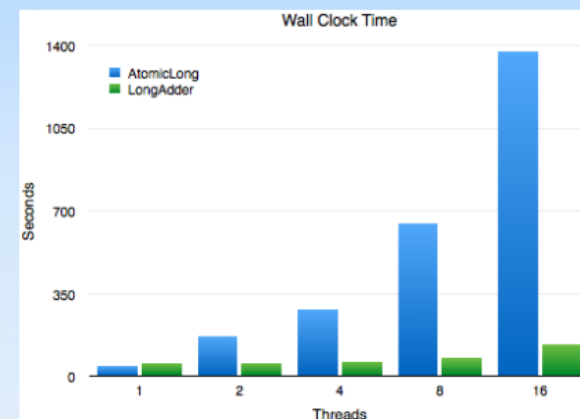
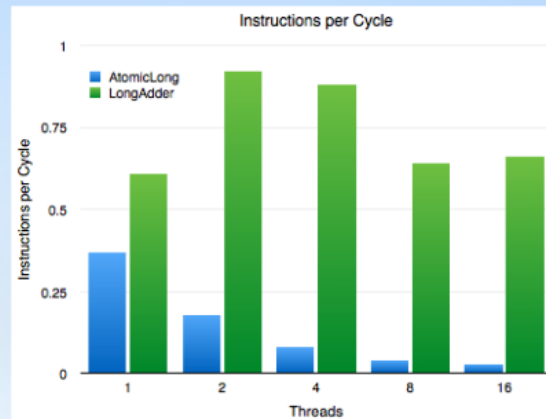
### Complex configuration!

```
akka {
  scheduler {
    implementation = akka.actor.LightArrayRevolverScheduler
    ticks-per-wheel = 32
  }
  actor {
    serialize-messages = off
    default-dispatcher {
      executor = thread-pool-executor
    }
  }
}
```

# Complex configuration!

```
akka {  
  scheduler {  
    implementation = akka.actor.LightArrayRevolverScheduler  
    ticks-per-wheel = 32  
  }  
  actor {  
    serialize-messages = off  
    default-dispatcher {  
      executor = thread-pool-executor  
      thread-pool-executor.core-pool-size-factor = 2.0  
      thread-pool-executor.core-pool-size-max = 1000  
      throughput = 128  
    }  
  }  
  ...  
}
```

# *Doom Upon All The World (AtomicXXX)*



Source: <http://blog.palominolabs.com/2014/02/10/java-8-performance-improvements-longadder-vs-atomiclong/>

- AtomicLong - all L1/L2 caches have to synchronise
- LongAdder - synchronises only on get
- -XX:+UseBiasedLocking
- Random.nextInt uses AtomicLong (use ThreadLocalRandom)

# ***Math vs. FastMath***

(org.apache.commons.math3.util.FastMath)

Name	StrictMath		FastMath		Math	
log	71	1.0	39	0.5555	21	0.2953
log10	94	1.0	108	1.1466	20	0.2174
log1p	74	1.0	103	1.3986	76	1.0269
pow	244	1.0	140	0.5726	238	0.9752
powII	159	1.0	120	0.7582	149	0.9415
exp	53	1.0	26	0.4959	39	0.7323
sin	42	1.0	34	0.8286	35	0.8464
asin	345	1.0	135	0.3911	347	1.0055
cos	46	1.0	37	0.8064	34	0.7414
acos	340	1.0	139	0.4079	354	1.0383
tan	80	1.0	68	0.8462	51	0.6461
atan	64	1.0	64	0.9931	66	1.0285
atan2	103	1.0	95	0.9257	104	1.0069
hypot	598	1.0	34	0.0573	592	0.9906
cbrt	95	1.0	67	0.7088	96	1.0151
sqrt	9	1.0	9	0.9818	9	1.0069
cosh	84	1.0	60	0.7248	85	1.0221
sinh	95	1.0	68	0.7159	98	1.0352
tanh	120	1.0	86	0.7193	117	0.9751
expm1	66	1.0	66	0.9961	66	1.0102
abs	1	1.0	5	2.8708	2	1.4429

# Summary

- CPU architecture matters and developer's desktop is not the same as HPC machine.
- System scheduler matters (a bit). CFS (Linux) vs. Multilevel Feedback Queue (Windows/Mac OS X)
- Jvm's children processes – check Context Switches '\_\_\_java' (pidstat -tw)
- Scala parallelism: 'par' collections vs. Akka Streams
- Atomic Operations vs. Adders
- Shop for libs: Math vs. FastMath

# Other Notes

- YourKit Java Profiler: <https://www.yourkit.com/java/profiler/features/>
- Scala Boxing/Unboxing: <https://stackoverflow.com/questions/6494860/how-to-spot-boxing-unboxing-in-scala>
- Java HotSpot VM Options: <http://www.oracle.com/technetwork/articles/java/vmoptions-jsp-140102.html>
- 'for' vs. 'while' loop: <https://stackoverflow.com/questions/16785826/why-are-scala-for-loop-comprehensions-so-very-slow-compared-to-for-loops>
- Specialised arrays: <https://www.scala-lang.org/old/node/10408.html>
- Java's ConcurrentSkipListMap: <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentSkipListMap.html>





# THANK YOU!

Questions?

