02: ♥♦ 13 Tips to write low latency applications in Java

Posted on October 26, 2015 by Arulkumaran Kumaraswamipillai

Extends Writing low latency applications in Java Q&A.

Tip #1: Use a **RTSJ** (Real Time Specification for Java) JVM. IBM, Oracle, and other smaller vendors have implemented this, but it comes at a cost. Oracle's **JavaRT**, IBM's **real-time WebSpere**, and aicas **JamaicaVM** to name a few popular ones. In real time JVM, instead of writing *java.lang.Thread* you just have to write, *javax.realtime.RealtimeThread*. Azul Zing is another JVM for a more predictable performance.

Tip #2: Big O notation for algorithms: Ensure all your data structures related algorithms are O(1) or at least O(log n). This is probably the biggest cause of performance issues.

Make sure that you have performance tests with real size data. Also, make sure that your algorithms are cache friendly. It is imperative to use proper cache strategies to minimize garbage collection pauses by having proper cache expiry strategy, using weak references for cache, reducing cache by carefully deciding what to cache, increasing the cache size along with the heap memory to reduce object eviction from cache, etc. Understanding Big O notations through Java examples

Tip #3: **Lock free algorithms**. Use lock free algorithms and non blocking I/Os. Even the most well designed concurrent application that uses locks is at risk of blocking. The **java.util.concurrent** package allows concurrent reads (e.g. ConcurrentHashmap). Lock free algorithms like CAS (Compare And Swap) can improve performance. 5 Java Concurrency interview Q&As.

The **Java NIO** (i.e.New I/O) supports non-blocking I/Os. 15 Java old I/O and NIO (i.e. New I/O) interview Q&As

Blocking is not good for low latency applications. Minimize context switching among threads by having threads not more than the number of CPU cores in your machine.

Tip #4 : **Reduce memory size**: Reduce the number of objects you create. Apply the flyweight design pattern where applicable. Favor stateless objects. Where applicable write immutable objects that can be shared between threads. Fewer objects mean lesser GC.

Tip #5: Tune your JVM: Tune your JVM with appropriate heap sizes and GC configuration. Before tuning profile your application with real life data. Basically you want to **minimize GC pause durations** and **increase GC throughput**. GC throughput is a measure of % of time not spent on GC over a long period of time. Specialist GC collectors like the Azul collector can in many cases solve this problem for you out of the box, but for many you who use the Oracle's GC, you need to understand how GC works and tune it to minimize the duration of the pauses. The default JVM options optimize for

throughput, and latencies could be improved by switching to the **Concurrent Garbarge** Collector.

GC tuning is very application specific. It is imperative to understand how your application uses the garbage collection. Memory is cheap and abundant on modern servers, but garbage collector pauses is a serious obstacle for using larger memory sizes. You should configure the GC to minimize the number & duration of the pauses.

- Enable diagnostic options (-XX:+PrintGCDetails -XX:+PrintTenuringDistribution -XX:+PrintGCTimestamps).
- Decide the total amount of memory you can afford for the JVM by graphing your own performance metric against young generation sizes to find the best setting.
- Make plenty of memory available to the younger (i.e eden) generation. The default is calculated from NewRatio and the -Xmx setting.
- Make the survival space to be same size as Eden (-XX:SurvivorRatio=1) and increase new space to account for growth of the survivor spaces (-XX:MaxNewSize= -XX:NewSize=)
- Larger younger generation spaces increase the spacing between full GCs. But young space collections could take a proportionally longer time. In general, keep the eden size between one fourth and one third the maximum heap size. The old generation must be larger than the new generation.

Tip #6:**Favor primitives to wrapper classes** to eliminate auto-boxing and un-boxing, especially in situations where the getter and setter methods are called very frequently for the wrapper classes like Integer, Float, Double, etc the performance is going to be adversely impacted due to auto boxing and unboxing. The operations like x++ will also provide poor performance if x is an ***Integer*** and not an **int**. So, avoid using wrappers in performance critical loops. Java primitives & objects – memory consumption interview Q&As

16 Technical Key Areas

Tip #7: **Good caching strategy and applying the short-circuit pattern**

Short-circuit pattern is handy for I/O related patterns like socket or URL based, database operations, and complex File I/O operations. I/O operations need to complete within a short amount of time, but with low latency Web sites, the **short-circuit pattern** can be applied to time-out long running I/O tasks, and then can either display an error message or the show **cached results**.

Tip #8: **Coding best practices to avoid performance issues due to death by 1000 cuts.**

- When using arrays it is always efficient to copy arrays using *System.arraycopy( )* than using a loop.
- When using short circuit operators place the expression which is likely to evaluate to false on extreme left if the expression contains &&.
- Do not use exception handling inside loops.
- Avoid using method calls to check for termination condition in a loop.
- Short-circuit equals( ) in large object graphs where it compares for identity first

```
1 @Override
2 public boolean equals(Object other) {
3     if (this == other) return true;
4     if (other == null) return false;
5
6     // Rest of equality logic...
7 }
8
```

Tip #9 : Experience and knowledge with some of the libraries like

- NIO-based scalable server applications by directly using **java.nio** package or framework like Apache MINA, Netty, Grizzly, etc.
- Actor based concurrency frameworks like **Akka**. Concurrency options in terms of increasing

abstractions: threads –> Executors –> Fork/Join –> Actor model.

- FIX protocol and commercial FIX libraries like **Cameron FIX**.
- Use Java 5 concurrency utilities, and locks.
- Lock free **Java disruptor library** for high throughput.
- **Chronicle Java library** for low latency and high throughput, which almost uses no heap, hence has trivial impact on GC.
- **Trove collection** libraries for primitives. Alternative for the JDK wrapper classes like *java.lang.Integer* for primitives requiring less space and providing better performance.
- **Javolution library** with real-time classes. For example, Javolution XML provides real-time marshaling and unmarshaling.

These libraries are aimed at providing reduced memory size, less impact on GC, lock free concurrent processing, data structure algorithmic efficiency, etc.

Tip #10: How is your data stored? Are you using a SQL database? How will that scale? Can you use a NoSQL data store instead. Transactional systems need SQL for transaction demarcation.

Relational and NoSQL data models are very different.

SQL Model:

The relational model takes data and store them in many normalized interrelated tables that contain rows and columns. Tables relate with each other through foreign keys. When looking up data, the desired information needs to be collected by joining many related tables and combined before it can be provided to the application.

NoSQL Model

NoSQL databases have a very different model. NoSQL databases have been built from the ground up to be

distributed, scale-out technologies and therefore fit better with the highly distributed nature of the three-tier Internet architecture. A document-oriented NoSQL database takes the data you want to store and aggregates it into documents using the JSON format. Each JSON document can be thought of as an object to be used by your application. This might relate to data aggregated from 10+ tables in an SQL model.

Tip #11: Pay attention to network round trips, payload sizes and type, protocols used, service timeouts and retries.

Tip #12: What volume of data are you dealing with? If you are using Gigabytes to terabytes of data, you are entering the space of big data and need to start thinking about

1) DFS (Distributed File Systems), e.g. HDFS (Hadoop Distributed File System) where your **250GB input file** can be split into 50 or more input splits and stored in 10 or more commodity hardware (normal cheap hardware systems). You can use cheap hardware because the same data is split and replicated across multiple systems as the **data nodes**. The "**name node**" keeps the meta data of what split data node is stored in which systems.

2) Map Reduce concept where "50 input splits" will have 50 **mappers** mapping data as **key/value** pairs on 50 different "data nodes". The mapped "key/value" data is passed to the "**reducers**" to aggregate the data and produce the **output files**. Unlike the number of mappers, the number of reducers will depend on the number of output files to be produced. If it produces 1 output file, then there will be only one "reducer".

Tip #13: WebSockets is a bidirectional communications channel as opposed to HTTP, which is unidirectional. Once the connection is established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode via a single socket (i.e. same TCP connection for the lifecycle of WebSocket connection). In HTTP, typically

a new TCP connection is initiated for a request and terminated after the response is received.

HTTP protocol is not only more verbose (e.g. require headers), but also either client can talk to server or server can talk to client, whereas WebSockets are duplex in nature allowing client and server to talk independent of each other. When one side closes the channel, the connection closes.

Popular Posts

♦ 11 Spring boot interview questions & answers

827 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

767 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

389 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

296 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

240 views

001B: ♦ Java architecture & design concepts interview questions & answers

202 views

Bio | **Latest Posts**

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ Spring, JavaConfig (@Configuration), and TransactionManager

07: Debugging Spring Transaction Management ›

Posted in Java Architecture Interview Q&A, Java Key Area Essentials, Judging Experience Interview Q&A, Low Latency, Performance, Scalability

Tags: TopX

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer