Are you ready

Friday, 2 November 2012

Power of Java MemoryMapped File

Power of Java MemoryMapped File

In JDK 1.4 interesting feature of Memory mapped file was added to java, which allow to map any file to OS memory for efficient reading. Memory mapped file can be used to developed IPC type of solution. This article is experiment with memory mapped file to create IPC.

Some details about Memory Mapped File, definition from WIKI

A memory-mapped file is a segment of virtual memory which has been assigned a direct byte-for-byte correlation with some portion of a file or file-like resource. This resource is typically a file that is physically present on-disk, but can also be a device, shared memory object, or other resource that the operating system can reference through a file descriptor. Once present, this correlation between the file and the memory space permits applications to treat the mapped portion as if it were primary memory.

Sample Program

There are two java program one is writer and other is reader. Writer is producer and tries to write to Memory Mapped file , reader is consumer and it reads message from memory mapped file. This is just a sample program to show to idea, it does't handle many edge case but good enough to build something on top of memory mapped file.

MemoryMapWriter

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class MemoryMapWriter {

 public static void main(String[] args) throws FileNotFoundException, IOException,
InterruptedException {
  File f = new File("c:/tmp/mapped.txt");
  f.delete();

  FileChannel fc = new RandomAccessFile(f, "rw").getChannel();

  long bufferSize=8*1000;
  MappedByteBuffer mem =fc.map(FileChannel.MapMode.READ_WRITE, 0, bufferSize);

  int start = 0;
  long counter=1;
  long HUNDREDK=100000;
  long startT = System.currentTimeMillis();
  long noOfMessage = HUNDREDK * 10 * 10;
  for(;;)
  {
   if(!mem.hasRemaining())
   {
    start+=mem.position();
    mem =fc.map(FileChannel.MapMode.READ_WRITE, start, bufferSize);
   }
   mem.putLong(counter);
   counter++;
   if(counter > noOfMessage )
    break;
  }
  long endT = System.currentTimeMillis();
  long tot = endT - startT;
  System.out.println(String.format("No Of Message %s , Time(ms) %s ",noOfMessage, tot)) ;
```

Search

Search

Java Code Geeks

Google+ Followers

About Me

Ashkrit

Pragmatic software developer who loves practice that makes software development fun and likes to develop high performance & low latency system.

View my complete profile

Blog Archive

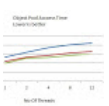
Popular Posts

Power of Java MemoryMapped File

Power of Java MemoryMapped File In JDK 1.4 interesting feature of Memory mapped file was added to java, which allow to map any file to O...

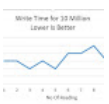
Which memory is faster Heap or ByteBuffer or Direct ?

Java is becoming new C/C++ , it is extensively used in developing High Performance System. Good for millions of Java developer like me:-) ...



Lock Less Java Object Pool

It is being while i wrote anything, i has been busy with my new job that involve doing some interesting work in performance tuning. One of ...



ArrayList Using Memory Mapped File

Introduction In-Memory computing is picking up due to affordable hardware, most of the data is kept in

```
 }

}
```

MemoryMapReader

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class MemoryMapReader {

 /**
  * @param args
  * @throws IOException
  * @throws FileNotFoundException
  * @throws InterruptedException
  */
 public static void main(String[] args) throws FileNotFoundException, IOException,
InterruptedException {


  FileChannel fc = new RandomAccessFile(new File("c:/tmp/mapped.txt"), "rw").getChannel();

  long bufferSize=8*1000;
  MappedByteBuffer mem = fc.map(FileChannel.MapMode.READ_ONLY, 0, bufferSize);
  long oldSize=fc.size();

  long currentPos = 0;
  long xx=currentPos;


  long startTime = System.currentTimeMillis();
  long lastValue=-1;
  for(;;)
  {

   while(mem.hasRemaining())
   {
    lastValue=mem.getLong();
    currentPos +=8;
   }
   if(currentPos < oldSize)
   {

    xx = xx + mem.position();
    mem = fc.map(FileChannel.MapMode.READ_ONLY,xx, bufferSize);
    continue;
   }
   else
   {
     long end = System.currentTimeMillis();
     long tot = end-startTime;
     System.out.println(String.format("Last Value Read %s , Time(ms) %s ",lastValue, tot));
     System.out.println("Waiting for message");
     while(true)
     {
      long newSize=fc.size();
      if(newSize>oldSize)
      {
       oldSize = newSize;
       xx = xx + mem.position();
       mem = fc.map(FileChannel.MapMode.READ_ONLY,xx , oldSize-xx);
       System.out.println("Got some data");
       break;
      }
     }
   }



  }

 }

}
```

Observation

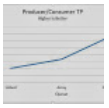
RAM to meet latency and throughput ...

Java Reflection Facts

Java has wonderful feature that allow to inspect any object at run time and extract useful information about it for e.g constructor, metho...

How To Write Micro benchmark In Java

So many article has been written on how to write micro-bench mark in java, this blog is my attempt to explain the topic. What is Micro b...

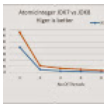


Executor With ConcurrentLinkedQueue

In this blog i will explore some of waiting strategy for inter thread communication. BlockingQueue is integral part of many concurrency f...

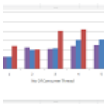
Java Queues - Bad Practices

Writing after long gap, looks like i lost the motivation or ran out of topic :-) Recently while going through code of my current assignme...



AtomicInteger Java 7 vs Java 8

Atomic Integer is interesting class, it is used for building many lock free algorithm. Infact JDK locks are also build using ideas from Ato...



Lock Free bounded queue

Lock free bounded queue JDK 1.5 was first step towards adding serious concurrency support to java, it changed the way you write concurre...

Memory mapped can be very good option for developing Inter Process communication, throughput is also reasonably well for both produce & consumer.

Performance stats by run producer and consumer together

Each message is one long number
Produce - 10 Million message - 16(s)
Consumer - 10 Million message 0.6(s)

Very simple message is used to show the idea, but it can any type of complex message, but when there is complex data structure then serialization can add to overhead. There are many technique to get over that overhead. More in next blog.

Posted by Ashkrit at 02:30

Labels: High throughput, Inter Thread communication, IPC, Low Latency, NIO

No comments:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)