

Author: Jon Barnett

Date: 24 July 2003

1 Parallel thread performance: A JBoss client example

A few questions have been raised about our thread performance tests. Although we have done the initial work to understand the impact of the environment on tests we were performing, we do not attempt to explain or explore all possible impacts of these observations. However, as we did notice these effects from some test work we were performing on raw throughput through JBoss, we will provide the samples we accumulated.

1.1 Test concept

Originally, we envisaged some simple tests on the name server and invocation of EJBs to check performance enhancements in these areas. The tests would queue remote lookup requests and remote method calls for the JBoss server and measure the time required to process a set number of requests. The load could be increased, given sufficient CPU, via parallel threads. The EJB used for the method invocation test would perform a low CPU intensive operation - incrementing a counter. Each client thread operates as an independent and parallel load, avoiding any explicit synchronisation or interaction with the parent or its peers.

In this configuration, there is minimal workload generated on the JBoss application server. With a multiprocessor system with a large number of CPUs, there should be a range of client thread quantities where the multiple parallel requests are able to be serviced by JBoss without there being an impact on the performance. That is; there are sufficient free CPU cycles such that parallel requests can be serviced in parallel.

A client thread serially makes requests to the JBoss server and does not make the next request until the current one has been serviced. When the client thread is waiting for a response it is not using the CPU. When a JBoss naming service thread or an EJB container thread is not generating a response it is not using the CPU. In theory, this would mean that the number of client threads could closely approach the number of available CPUs before saturation effects would affect performance. As CPU sharing increases, there is expected to be degradation in throughput, probably resulting in an exponential increase in completion times for tasks.

Inverting the original aim of the test, the effect of parallel thread management may be observed in the test configuration. From our previous investigation in [Java thread performance](#) we have seen that parallel thread performance suffers as the number of threads increase. This effect might also be visible in the results from this test, given sufficient CPUs. The test can also indicate possible performance behaviour of a multi-threaded JBoss client. The final benefit of this test is to observe the performance of a complex client interaction in a parallel thread load.

1.2 Test environment

The test environment consists of an 8-way Intel multiprocessor system, running RedHat 9 with an NPTL implementation. Nothing else is running on the system except the standard Linux support processes. The CPU clock speed on the system is reported as 2.0 GHz. The bus clock speed is reported as 100.0 MHz. JBoss 3.2.0 runs within an IBM Java 2 SDK 1.4.1 build cxia32141-20030522 Java Virtual Machine (JVM) with LD_ASSUME_KERNEL=2.2.5. This JVM configuration provides a known performance characteristic without non-linear effects under normal conditions. This ensures that the client response is not excessively affected by server-side operations.

1.2.1 Java Virtual Machine environments

The following JVM client environments are tested:

- Sun Java 2 JDK 1.4.1 03 b02 in server mode
- IBM Java 2 SDK 1.4.1 build cxia32141-20030522 with LD_ASSUME_KERNEL=2.2.5
- Sun Java 2 JDK 1.4.2 b28 in server mode with -XX:+UseNewParGC
- Sun Java 2 JDK 1.4.2 b28 in server mode with -XX:+UseNewParGC and -XX:+UseConcMarkSweepGC

The LD_ASSUME_KERNEL=2.2.5 setting instructs the IBM SDK to employ the standard Linux threading model rather than use NPTL.

The IBM Java 2 JVM cannot be used in NPTL mode because the client program fails to terminate. As the Sun Java 2 JDK 1.4.2 has a similar characteristic to the Sun Java 2 JDK 1.4.1 with standard GC settings, we have not tested this mode in JDK 1.4.2 specifically. The reader may conduct this test themselves to confirm our assumption if they so wish.

1.3 Name lookup test

This test creates client threads that continually request a name lookup from the JBoss 3.2.0 server. The duration for each thread is measured over 10000 requests.

1.3.1 Remote lookup client code

The code below represents the client thread used in the test.

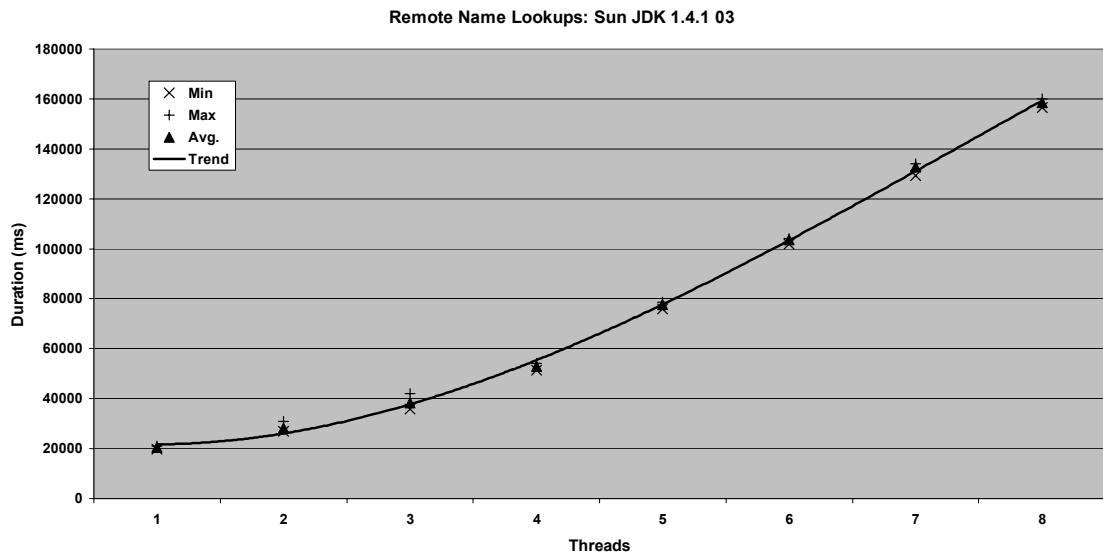
```
class ServiceRemoteLookupTest extends Thread
{
    private int id = 0;
    private Properties jndiProps = null;

    public ServiceRemoteLookupTest(int id, Properties jndiProps)
    {
        this.id = id;
        this.jndiProps = jndiProps;
    }

    public void run()
    {
        TesterHome home;
        Object reference;
        Context naming = null;
        long counter = 0;
        try
        {
            naming = new InitialContext(jndiProps);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        Date start = new Date();
        for (int i = 10000; i-- > 0; )
            try
            {
                {
                    reference = naming.lookup("Tester");
                    home = (TesterHome) PortableRemoteObject.narrow(reference,
                        TesterHome.class);
                }
                catch(Exception e)
                {
                    e.printStackTrace();
                }
            }
            Date end = new Date();
            counter = end.getTime()-start.getTime();
            System.out.println("Remote lookup " + id + ": " + counter + " ms");
        }
    }
}
```

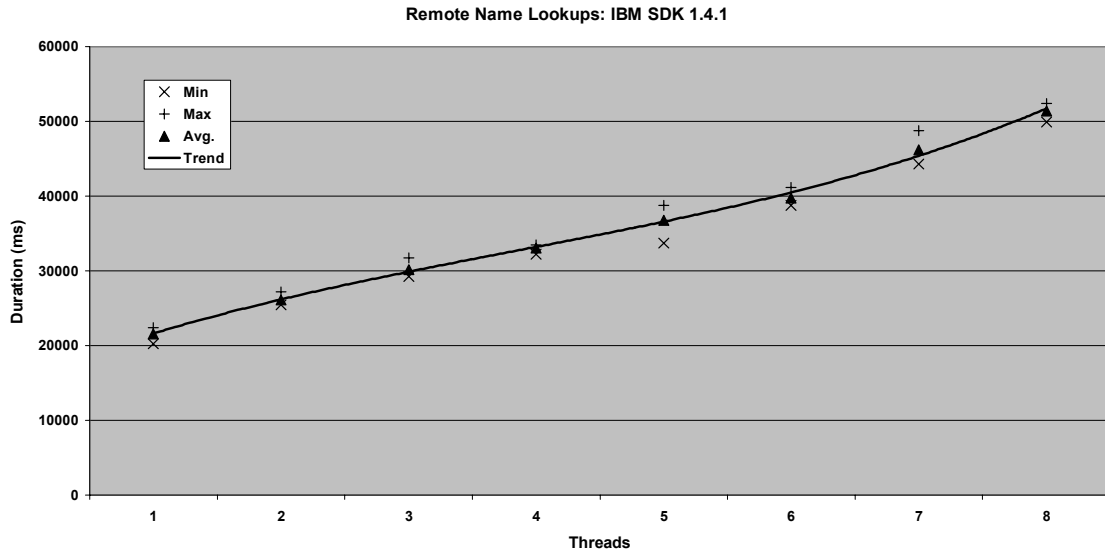
1.3.2 Sun JDK 1.4.1 03 results

Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	19659	20944	20624	315
2	16	26879	30828	28284	1177
3	15	35809	42010	38257	1970
4	16	51445	54079	53028	960
5	15	75877	78739	77633	888
6	18	101900	104116	103521	572
7	21	129258	134128	132734	1227
8	16	156433	159852	158386	1025



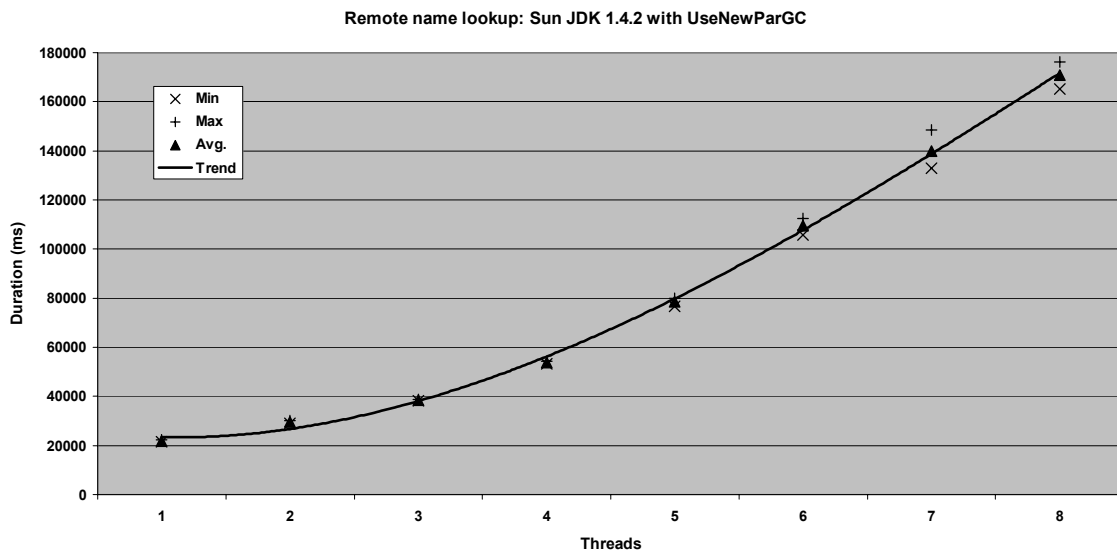
1.3.3 IBM SDK 1.4.1 results

Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	20209	22438	21598	708
2	16	25461	27226	26127	489
3	15	29231	31734	30185	870
4	16	32253	33450	33017	369
5	15	33697	38724	36737	1972
6	18	38796	41130	39780	851
7	21	44264	48765	46227	1768
8	16	49957	52372	51436	700



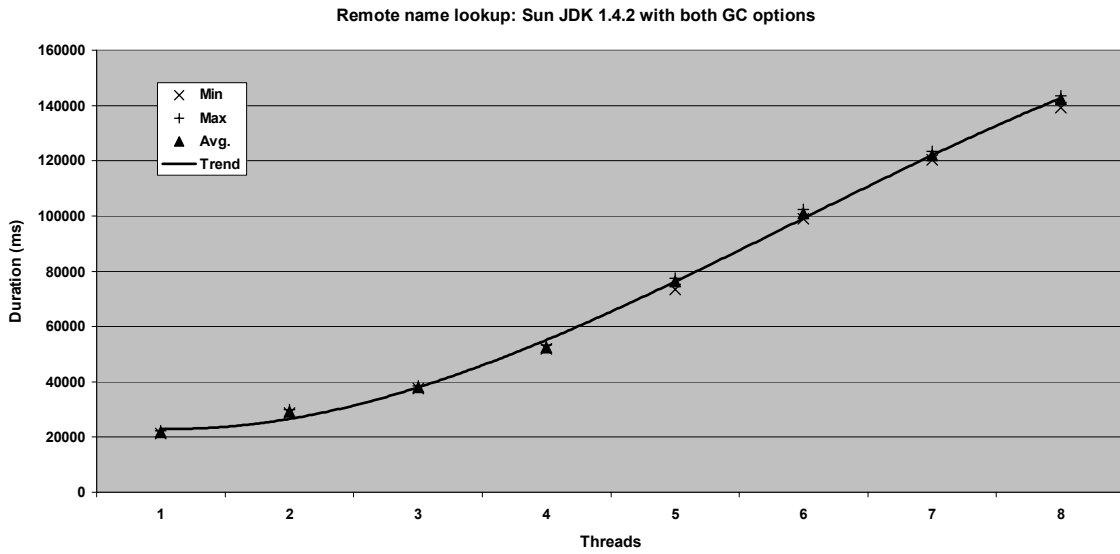
1.3.4 Sun JDK 1.4.2 UseNewParGC results

Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	21549	22200	21894	178
2	16	29019	30069	29584	351
3	15	38308	38856	38599	156
4	16	53285	54389	53692	327
5	15	76809	80037	78536	967
6	18	105616	112393	109411	1931
7	21	132796	148511	139924	5897
8	16	165304	176168	170722	5014



1.3.5 Sun JDK 1.4.2 new GC results

Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	21218	22296	21849	284
2	16	28651	29823	29202	372
3	15	37497	38544	38012	344
4	16	51793	53384	52423	558
5	15	73325	77542	76260	1021
6	18	98968	102325	100829	1028
7	21	120247	123346	121882	955
8	16	139305	143395	142260	1028



1.4 Remote invocation test

This test creates client threads that continually invoke a method from a stateless session bean on the JBoss 3.2.0 server. The duration for each thread is measured over 10000 requests. The naming server lookup is excluded from the measurement.

1.4.1 Remote invocation client code

The code below represents the client thread used in the test.

```
class ServiceRemoteTest extends Thread
{
    private int id = 0;
    private Properties jndiProps = null;

    public ServiceRemoteTest(int id, Properties jndiProps)
    {
        this.id = id;
        this.jndiProps = jndiProps;
    }

    public void run()
    {
        TesterHome home = null;
        long counter = 0;
        try
        {
            {
                Context naming = new InitialContext(jndiProps);
                Object reference = naming.lookup("Tester");
                home = (TesterHome)PortableRemoteObject.narrow(reference,
                    TesterHome.class);
            }
        }
    }
}
```

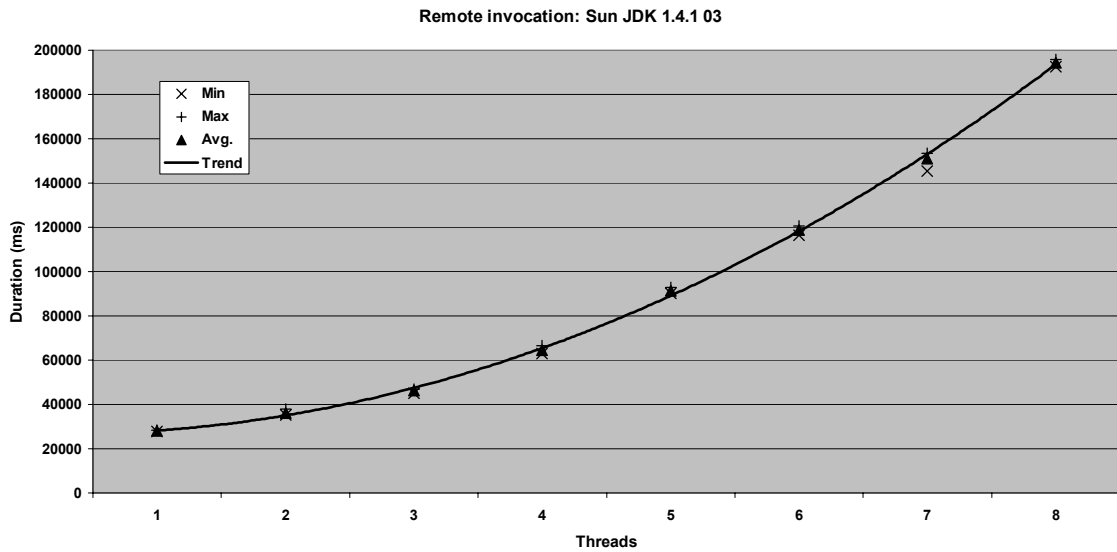
```

catch(Exception e)
{
    e.printStackTrace();
}
Tester tester;
Date start = new Date();
for (int i = 10000; i-- > 0; )
    try
    {
        tester = (Tester)home.create();
        counter = tester.serviceTest(counter);
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
Date end = new Date();
counter = end.getTime()-start.getTime();
System.out.println("Remote " + id + ": " + counter + " ms");
}
}

```

1.4.2 Sun JDK 1.4.1 03 results

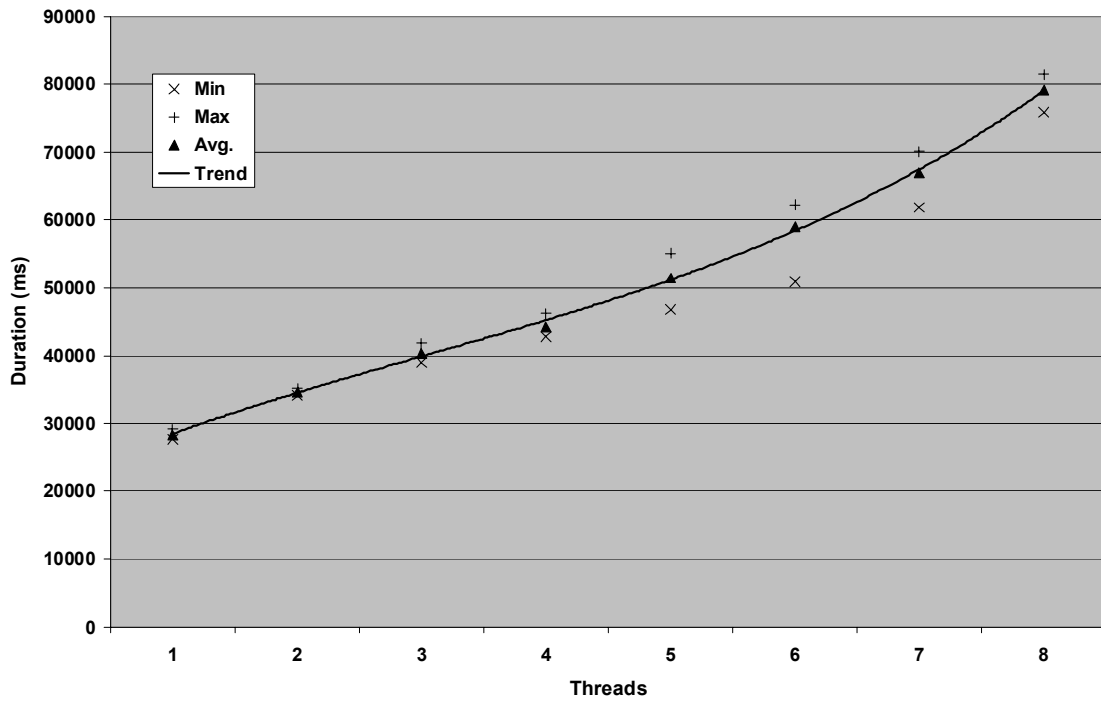
Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	27650	28390	28020	211
2	16	35286	37926	36121	782
3	15	45096	46826	46163	556
4	16	62888	66463	64406	1218
5	15	90117	92699	91263	944
6	18	116561	120514	118746	1082
7	21	145424	153561	151186	2218
8	16	192614	195736	194320	996



1.4.3 IBM SDK 1.4.1 results

Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	27708	29194	28376	439
2	16	34099	35224	34694	353
3	15	38983	41754	40311	672
4	16	42778	46197	44275	1178
5	15	46753	55049	51455	2364
6	18	50895	62240	58997	2663
7	21	61807	70026	66966	2140
8	16	75869	81382	79145	1630

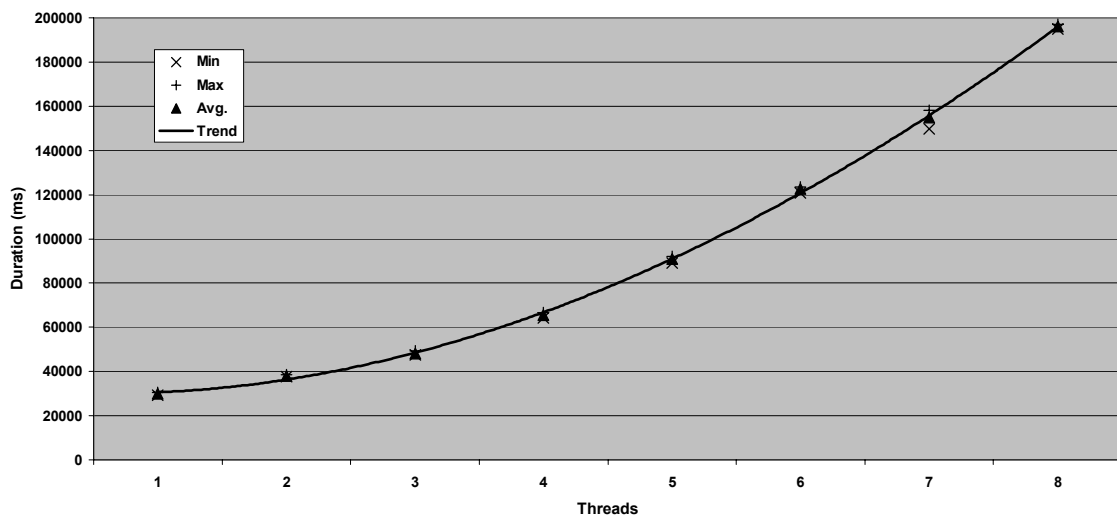
Remote invocation: IBM SDK 1.4.1



1.4.4 Sun JDK 1.4.2 UseNewParGC results

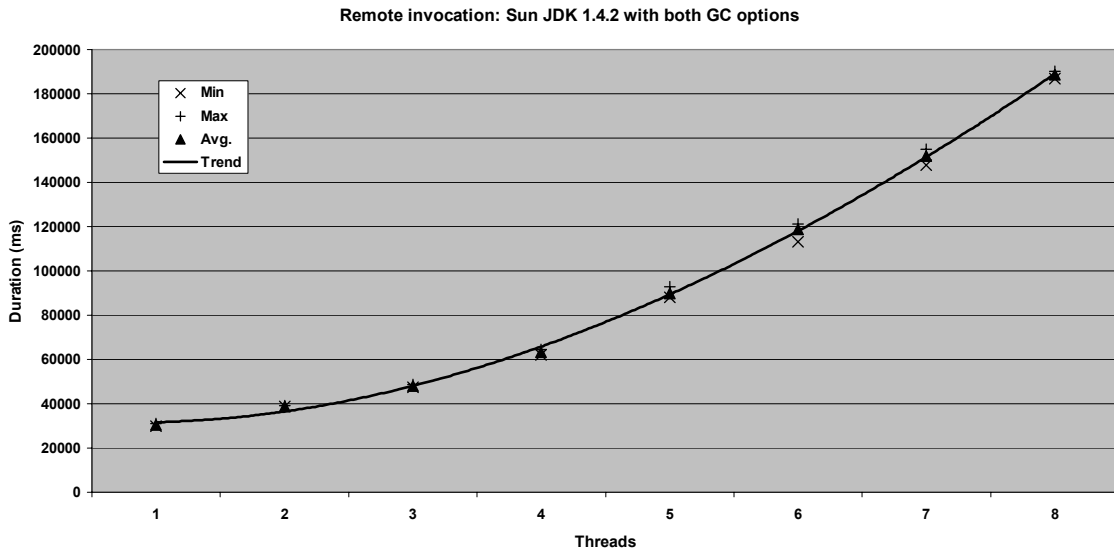
Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	29363	30545	29922	367
2	16	37557	38543	38085	323
3	15	47427	49353	48013	542
4	16	64273	66607	65334	710
5	15	89163	92067	90968	863
6	18	120807	123471	122470	921
7	21	149777	158007	154721	2245
8	16	195022	196874	196053	553

Remote invocation: Sun JDK 1.4.2 with UseParNewGC



1.4.5 Sun JDK 1.4.2 new GC results

Threads	Samples	Min.	Max.	Avg.	Std. Dev.
1	16	29865	30914	30591	327
2	16	38744	39168	38949	148
3	15	47371	48685	47993	390
4	16	62145	64493	63374	612
5	15	87921	92764	89852	1954
6	18	113106	121114	118802	1947
7	21	147893	154808	151944	1925
8	16	186833	190069	188575	1216



1.5 Discussion

The IBM SDK responses for both remote lookups and remote invocation follow theory in that the responses suffer from degradation as the number of threads approach the maximum number of CPUs. These are very slight deformations in the response characteristic indicating that both the server and the clients are not suffering unduly from CPU contention. The remote invocation shows greater signs of CPU contention, but is within expectation as the JBoss server is performing more work than with the simple naming server lookup. Other than that, the linear characteristics corroborate our original performance test results.

The Sun JDKs demonstrates the characteristics of a second order polynomial for both responses and all modes. The response characteristics either smoothly incorporate the deformation due to CPU contention or the natural bottleneck of the thread management overhead is preventing any CPU contention from occurring. The curves characteristics help confirm the results of our previous investigations.

These results also indicate our original thread load model that included object creation may be the more prevalent code mix rather than our simplified load model without the object creation operation. This will only be established with more field reports. It also shows that in this instance, the simple enablement of the new GC algorithms is not enough to streamline the complex client operation. The latest Sun JDK provides a large set of tuning parameters, but requires tuning requires an understanding of the detailed operation of programs. It is still too early to determine if there are metrics to help simplify tuning of the JVM for application server loads. Application servers may experience a wide variety of load characteristics and so the JVM may require an amount of hand-tuning, based on the results we obtain from this testing.

The response deformation for the Sun JDK that we have recorded here confirms the results of our initial parallel thread investigation. However, it does not provide any indication of operation in a production situation. Although there is a suggestion that the management overhead of parallel Java threads detracts from the benefit of additional CPU for larger CPU numbers, there is insufficient field evidence to substantiate the results of these investigations and there is not enough information at this time to determine the source of the performance degradation.

As always, these results are provided as an informational note. They should be used as a guide to physical effects that may influence outcomes in your own Java test and production environments. However, they should not be used as an absolute indication of performance as that will depend on many factors and interactions beyond the scope of this study to take into account.

In order to aid the visual interpretation of the results, we direct you to read the technical note, [Parallel thread performance: A visual aid](#).