SPECjbb2005

Alan Adamson, IBM Canada
David Dagastine, Sun Microsystems
Stefan Sarne, BEA Systems

Topics

- Benchmarks
- SPECjbb2000
  - Impact
- Reasons to Update
- SPECjbb2005
  - Development
  - Execution

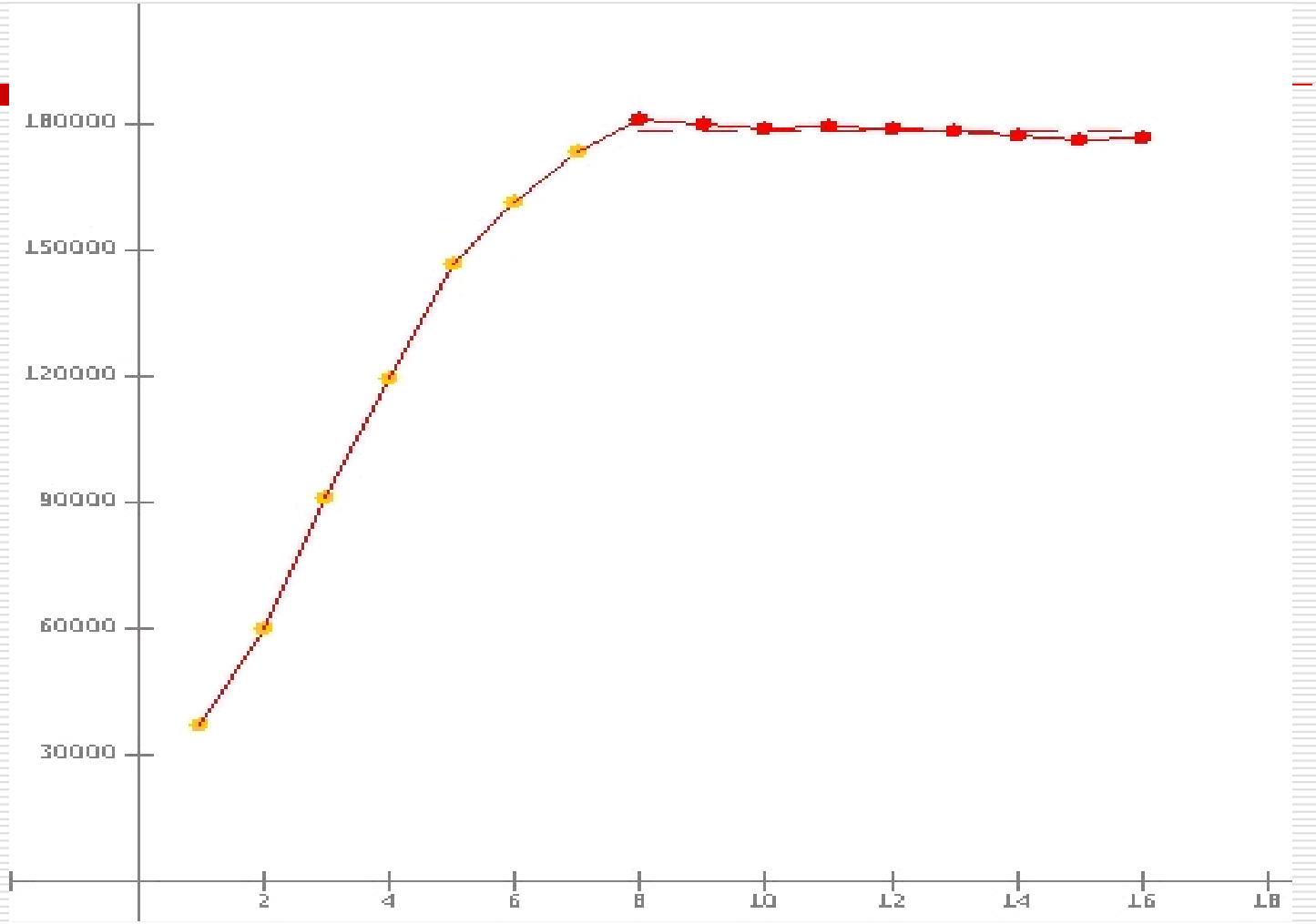
Benchmarking

- Uses of benchmarks
  - Estimation of system requirements (use for the customer), choice of a candidate supplier
  - Target for optimization efforts (use for the vendor, or possibly a researcher)
    - Evaluation of release-to-release improvements
    - Evaluation of impact of proposed prototyped ideas
  - Performance marketing (should be useful to both customer and supplier)
- Performance marketing
  - Usually "system x runs better on code y than system z"
  - Having two many potential "code y"'s around is tough for a development group
    - Do not know how you will be compared
    - So do not know what to work on
  - And maybe for customers
    - Do not know the value of a benchmark

SPECjbb2000

- Based on IBM code designed to test some C++ components
- Based on TPC-C workload
- Written in Java, using a persistence framework that was originally written in C++
- Transactional, and highly parallel
  - Database in-memory, using that persistence framework
  - No file i/o or network activity measured
- Run transactional load from 1 to X warehouses (threads doing same load)
- For each warehouse number, 30 second warmup, and a 2-minute measurement period – record the number of transactions
- Score is the mean value from the warehouse with the peak value to twice that
- Simple install-and-go benchmark, popular for Java performance analysis – get a pretty scaling graph

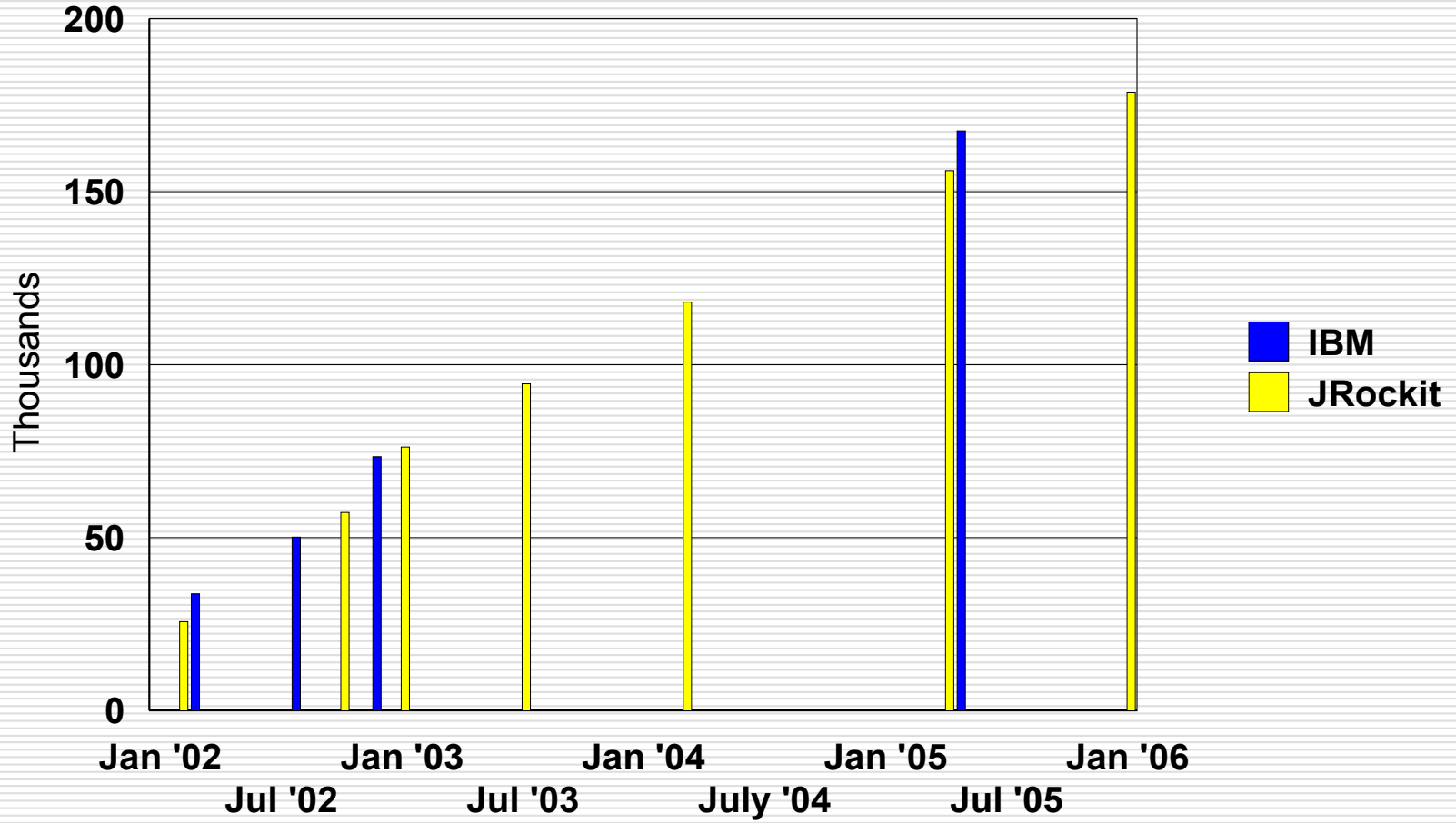
Typical report



Impact

- Active publication history
  - 366 (24-58-57-52-39-122-14)
- Significant impact on JVM Technology
  - Refinements on locking
  - GC challenged by large heaps
  - Initial lead score of 80,348 progressed to 2,505,420 in about 5 years

SPECjbb2000 – 4-Core Intel Systems leading at publication



Problems with SPECjbb2000

- Not very 'java'-ish
  - Much is transliterated C++
  - Uses roll-your-own data structures rather than java collections
  - Overall structure not very object-oriented
- Financial calculations in float
  - Should use BigDecimal
- No XML processing
- No standard logging (java.util.logging)
- Odd 'fairness' requirement
  - 'thread spread'
  - Exposed by Intel hyperthreading (and strong affinity scheduling)
- Unrealistically parallel (much unnecessary synchronization)

SPECjbb2000 and large systems

- Run-time
  - Roughly 2.5 mins per warehouse
  - Warehouses from 1 to 2*number of threads – could be 256
  - 640 minutes = 10 hours +
- Hit peak at 'wrong place'
  - Lots of jitter in results at the high end (GC, threading)
- Hard limit of 255 warehouses (scores for later whs count for 0)
- Garbage collection and very large heaps
  - System.GC() called between measurement periods
  - Allowed a generational GC strategy for large systems with JREs for which an old space GC could be a catastrophe.
  - Not a realistic customer scenario

General Goals

- Maintain a similar basic workload
  - Well understood, easy to analyze, easy to run (load-and-go)
  - Keep the pretty graphs
- Maintain same target scope – single address–space
  - Shared memory systems
- Try to use java libraries wherever possible
  - Put pressure on development teams to improve the libraries
  - Not historically a major focus of interest from development teams
    - Literature largely about JIT, GC
- Try to maintain a reasonably realistic usage scenario
- Simplify the process of running and submitting the benchmark
- Create a Java 5.0 benchmark

Specific Goals

- Replace the persistence framework with uses of collection classes
- Introduce BigDecimal for monetary calculations
- Add standard JSE Logging
- Introduce XML usage
  - Display (old green screens)
  - Use XML messages in a queue to distribute work
- Get rid of System.GC()s – with implications
  - Longer measurement period
- Multiple JVM option?

How Did We Do?

- Make it more java-ish
  - Removed the persistence framework
  - Replaced the previous data structures by HashMaps and TreeMaps
  - Re-factored code for more object orientation (and along the way, introduced use of Interfaces)
  - Overall a success
- Make it Java5
  - Quite successful – generics, auto-boxing, a few others
- Financial calculations in float
  - Changed all monetary uses of float to BigDecimal
  - This hits performance very significantly – not just in the computation time, but the allocation load as well
  - We have all adapted
- No XML Processing
  - Re-wrote the display screen processing to build a DOM (Sun)
  - (but XML gets written but never read, so no parsing is exercised)

How Did We Do? - 2

- No standard logging
  - Introduced logging via use of java.util.logging
  - Helpful tool
- Mitigate the complexity of the thread spread requirement
  - Dropped the requirement
  - Archives did not explain the rationale, and nobody could figure out the justification
- Unrealistically parallel
  - Proposed and prototyped a queuing mechanism whereby transactions for all processors would be scheduled via shared work queues, with transaction requests as XML packets (BEA)
  - Performance could not be made reasonable so this was abandoned
  - Means there is a lot of uncontended locking

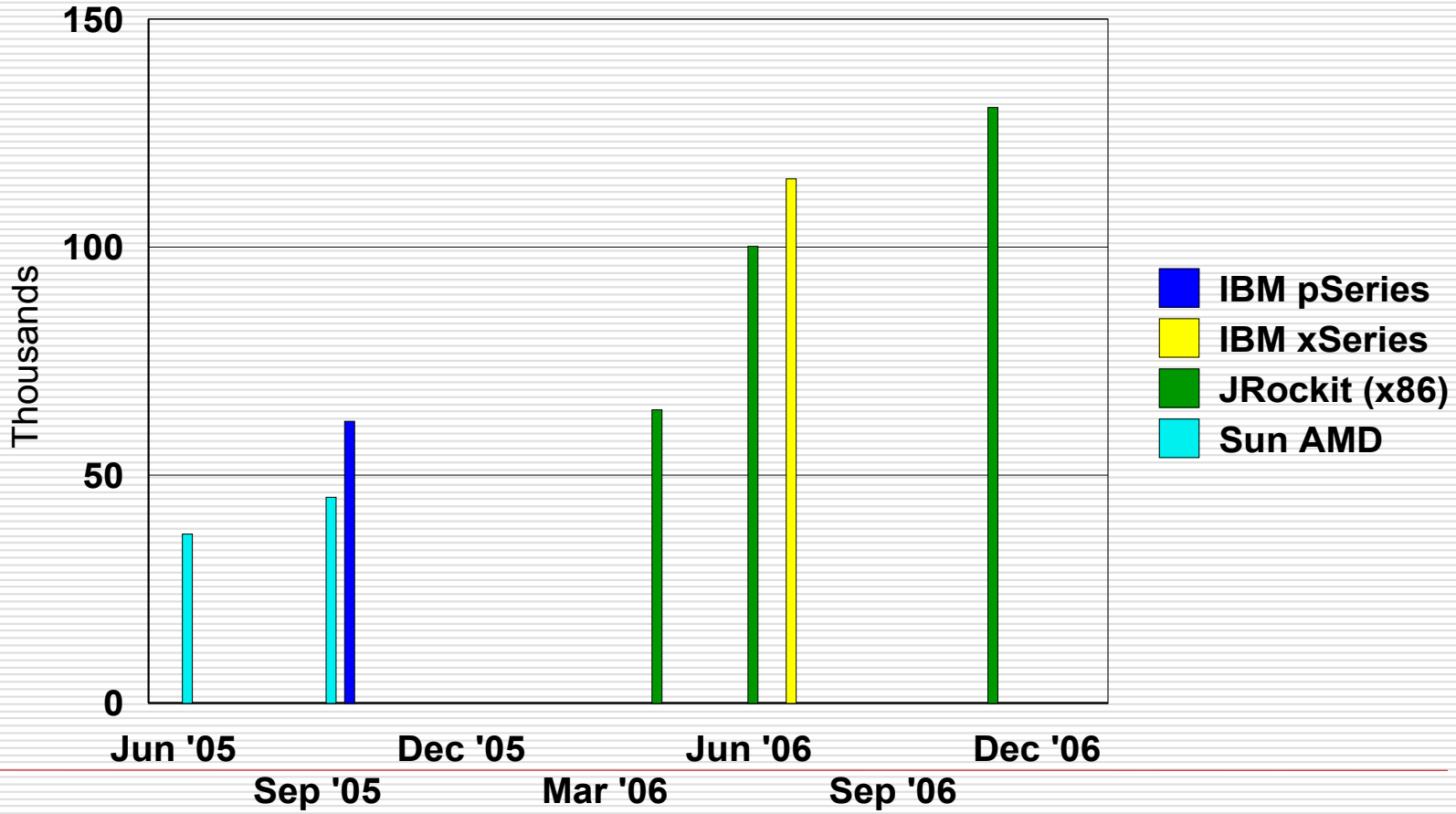
How Did We Do? - 3

- GC
  - Eliminated the System.GC()s – excellent
- High allocation rate
  - A realistic pressure point
  - Too concentrated in hot routines
- Run-Time on Large Systems
  - Multi-JVM – e.g. on a 64-way, run 4 JVMs simultaneously as if each was on a 16-way (cutting runtime by 4)
  - Controversial
    - No longer testing VM scaling, but some combination of OS and VM scaling
    - But a 'realistic' model, as app server deployments are often multi-JVM
    - Concerns largely addressed by having 2 metrics, SPECjbb2005 bops, and SPECjbb2005 bops/JVM, and both must be stated
- Mixed result – maybe should have had tighter run rules to separate the two types of run
  - Being used on small systems, possibly to hide scaling problems
  - Also to mitigate NUMA hardware characteristics

Eighteen Months of SPECjbb2005

- Much leapfrogging of key vendor results
  - Overall and category leads passing back and forth
- Very significant JRE changes (largely visible on command-lines) with giant impact
  - Biased locking (with other names)
  - Simple BigDecimal optimization
  - Other library work (HashMap, etc)
  - Others we don’t know the other guy has done
  - General impact likely useful in wider cases
- Progress
  - 4-core from 37,034 to 130,589 SPECjbb2005 bops
  - 2-socket from 24,208 to 210,065 SPECjbb2005 bops

SPECjbb2005 History – 4-Core Systems leading at publication



Disclaimers

- SPEC and SPECjbb are registered trademarks of the Standard Performance Evaluation Corporation
- SPECjbb2000 results were leading 4-core Intel-based results since early 2002
- SPECjbb2005 results were 4-core results leading at the time of publication
- All results cited are results at www.spec.org as of January 21, 2007