Characterizing a Java Implementation of TPC-W

Todd Bezenek, Trey Cain, Ross Dickson, Tim Heil, Milo Martin, Collin Mccurdy, Ravi Rajwar, Eric Weglarz, Craig Zilles, and Mikko Lipasti

Department of Electrical and Computer Engineering
Computer Sciences Department
University of Wisconsin-Madison

Outline

- What is TPC-W?
- Our implementation of TPC-W
  - Why Java?
- Full system simulation
- Results
- Future work and summary

What is TPC-W?

- TPC-W is the TPC's **newest benchmark**
  - **Version D5.5** (11/19/99), final version due 1Q 2000
  - Our implementation is based on vD5.5
  - www.tpc.org
- Measures systems for **transactional web environments**
- Transactional web environment
  - **Web serving** of static and dynamic content
  - On-line transaction processing (**OLTP**)
  - Some decision support (**DSS**)

What does TPC-W model?

- Models an **online bookstore**
  - Searching
  - Browsing
  - Shopping carts and secure purchasing
  - Best sellers and new products
  - Customer registeration
  - Administrative updates

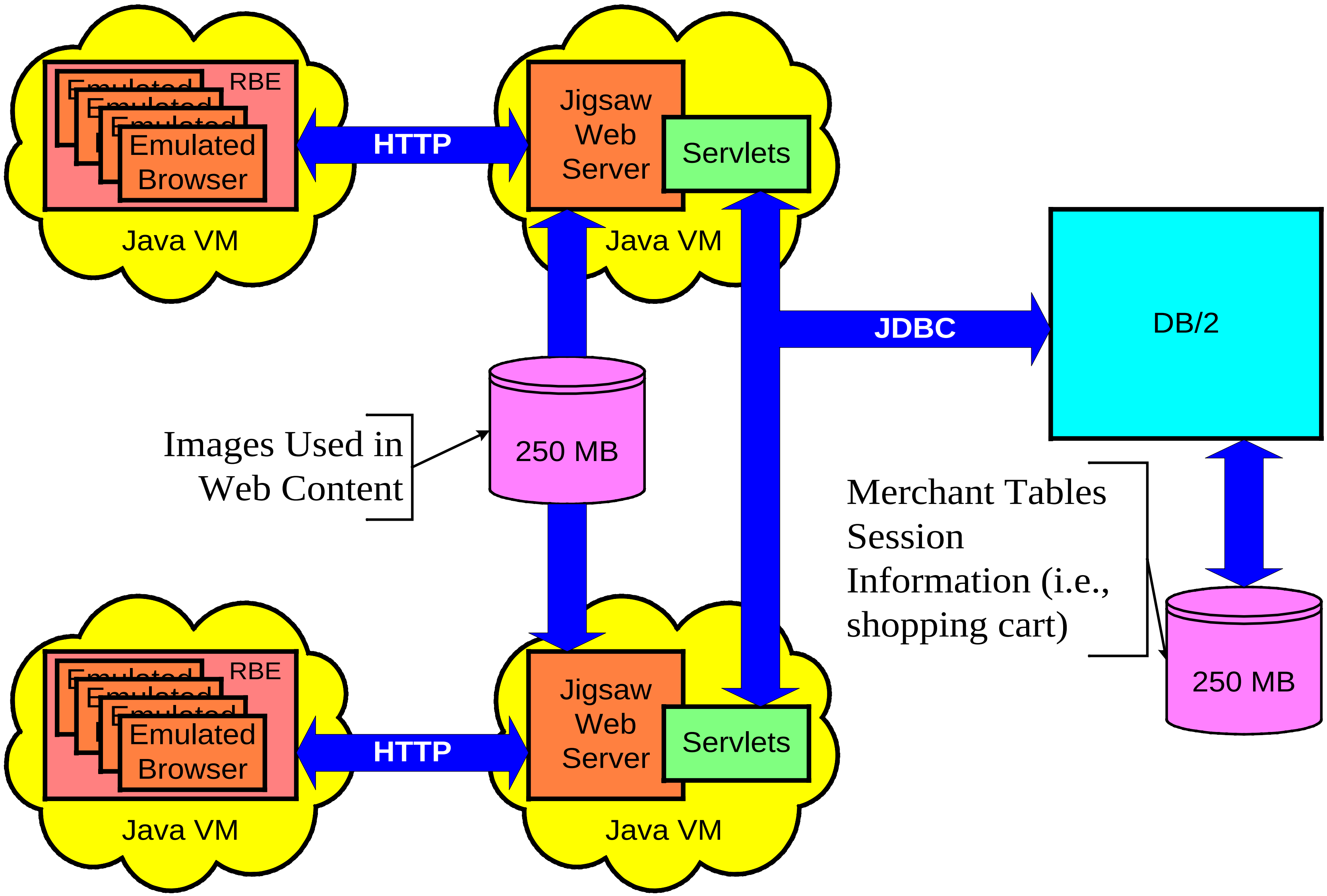
Observations about TPC-W

- **Dynamic web pages**, static images
- **Durable shopping cart**
- **Lazy consistency**
  - Allows 30 seconds for some pages to be updated
  - Enables various caching optimizations
  - We did not exploit this opportunity
- **Scaling**
  - ~5MB in DB tables per concurrent user (like TPC-C)
  - ~1KB per item in DB tables (like TPC-D)
  - ~25KB per item in static images

Our TPC-W implementation

- **All 14 web interactions** implemented
- Components
  - Jigsaw **Java web server** (www.w3.org/Jigsaw)
  - Server side Java '**servlets**'
  - Java Database Conectivity (**JDBC**)
  - IBM's **DB2** Universal Database 6.1
  - **Images stored in filesystem**
- **Did not implement**
  - Secure sockets layer (**SSL**)
  - Payment gateway emulator (**PGE**)

Our TPC-W implementation



Why Java?

- **Portability**
  - Studied workload on PowerPC and Sparc ISAs
  - Workload ran on **both architectures with no changes**
- **JDBC interface**
  - Connecting server side applications to a database
  - **Simple and elegant**
- **Server side Java**
  - Java servlet behavior not well understood
  - Opportunity to study new environment

Why full-system simulation?

- TPC-W has
  - **Multiple users, threads, and components**
  - **Significant inter-process communication**
  - TCP/IP **networking**
  - **File caching** of static content

- **Performance counters are not enough**

- **Full-system simulation is necessary** for accurate and complete characterization of TPC-W

- Simulated two architectures

SimOS-PowerPC

- **AIX 4.3.1** (slightly modified)
- 64-bit **PowerPC ISA**
- **Simulates device interfaces** → modified device drivers
- **Checkpointing support**
- Fast simulation through **runtime code-generation**
  - Runs only on AIX PowerPC machines
- **Emitter interface** for trace-based studies
- **Source code available**
- SimOS-PPC publicly available

SimICS

- **Solaris 7** (unmodified)
- 64-bit **Sparc v9 ISA**
- **Simulates hardware devices** → unmodified drivers
- Fast simulation through threaded code and **simulator translation cache** (STC)
- Source code not generally available
  - Runs only on Solaris/SPARC machines
  - **Add code through loadable modules**
- **www.simics.com** (Virtutech)

Full-system simulation challenges

- Significant work in getting the **right disk image**
- External ethernet & TCP/IP **networking simulation tricky**
  - Machine-room simulation?
- **Checkpointing simulated multi-tier implementation**
  - External interactions add complexity
- **Simulation speed/detail tradeoff**
- Large workload requires simulating large systems
  - Multiple processors
  - Large memories
  - Long warmups

Simulation parameters

- **Single-tier** configuration
  - **All** servers & browser emulators **on single system**
  - One web server
  - **Eight emulated browsers** with no think time
- Dual processor (SimICS), Uniprocessor (SimOS)
- **1 GB main memory, Single-level cache**
- **~250 MB** of **database** tables
  - 144000 customers, 10,000 items
- Images not served
- Database warm-up by full table scans on all tables
- JVM 1.1.x (No JIT)

Workload CPU breakdown

SimICS (from Unix utility: *top*)

Process	System Utilization
rbe	2 - 5 %
jigsaw	15 - 25 %
db2	70 - 83 %

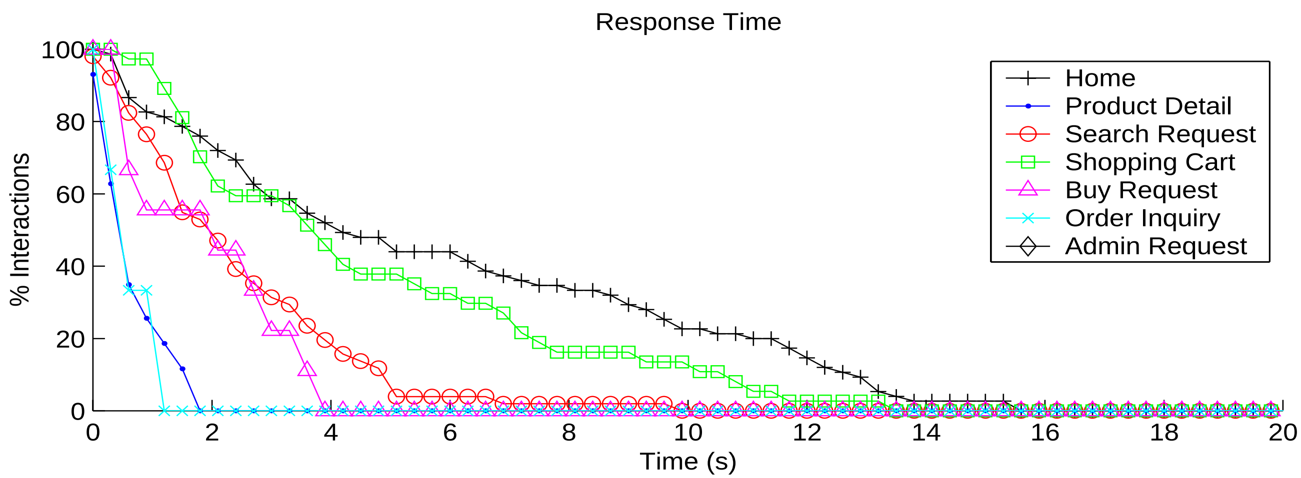
In kernel mode: 5 -15 %

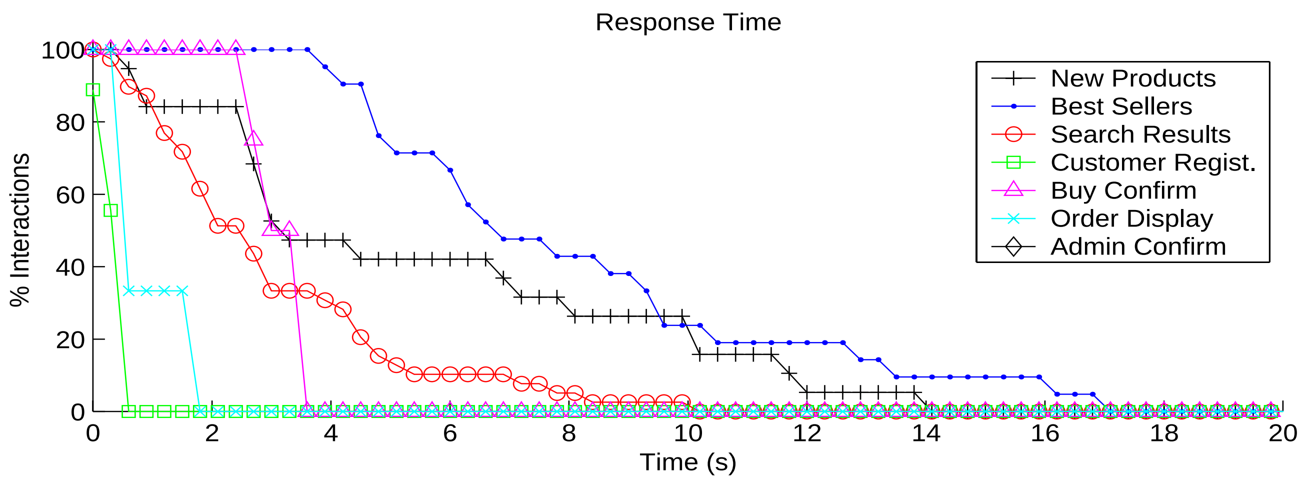
SimOS-PPC (from emitter interface)

Process	Instructions	Loads	Stores
kernel (1 thread)	33 %	23 %	23 %
rbe (10 threads)	5 %	2 %	5 %
jigsaw (14 threads)	23 %	16 %	22 %
db2 (17 threads)	38 %	59 %	49 %

Response time (from a real system)

Searches and ‘best sellers’ requests dominate database server utilization





Instruction supply (SimICS/SimOS)

SimICS:

I-Cache Hit Rate (150M inst.)
64-byte block, 4-way

Size	Proc#1	Proc#2
4KB	91.9 %	92.6 %
16KB	94.4 %	95.0 %
64KB	97.5 %	97.7 %
256KB	99.0 %	99.1 %
1MB	99.7 %	99.7 %

Overall Branch Predictor Accuracy

Predictor	Proc#1	Proc#2
small 2-bit (128B table)	84.1 %	85.1 %
medium 2-bit (1KB table)	93.4 %	92.8 %
large 2-bit (16KB table)	96.6 %	95.5 %
small gshare (10b history, 256B table)	89.5 %	89.2 %
large gshare (16b history, 16KB table)	96.7 %	95.4 %

SimOS-PPC:

I-Cache Hit Rate (2.5B inst.)
64-byte block, 2-way

Size	Proc#1
4KB	95.6 %
16KB	97.7 %
64KB	99.1 %
256KB	99.7 %
1MB	99.9 %

Per Thread Branch Predictor Accuracy
large gshare (16b history, 16KB table)

Thread	Accuracy
kernel	96.7 %
rbe	96.4 %
jigsaw	93.6 %
db2	93.0 %

Data supply (SimOS/SimICS)

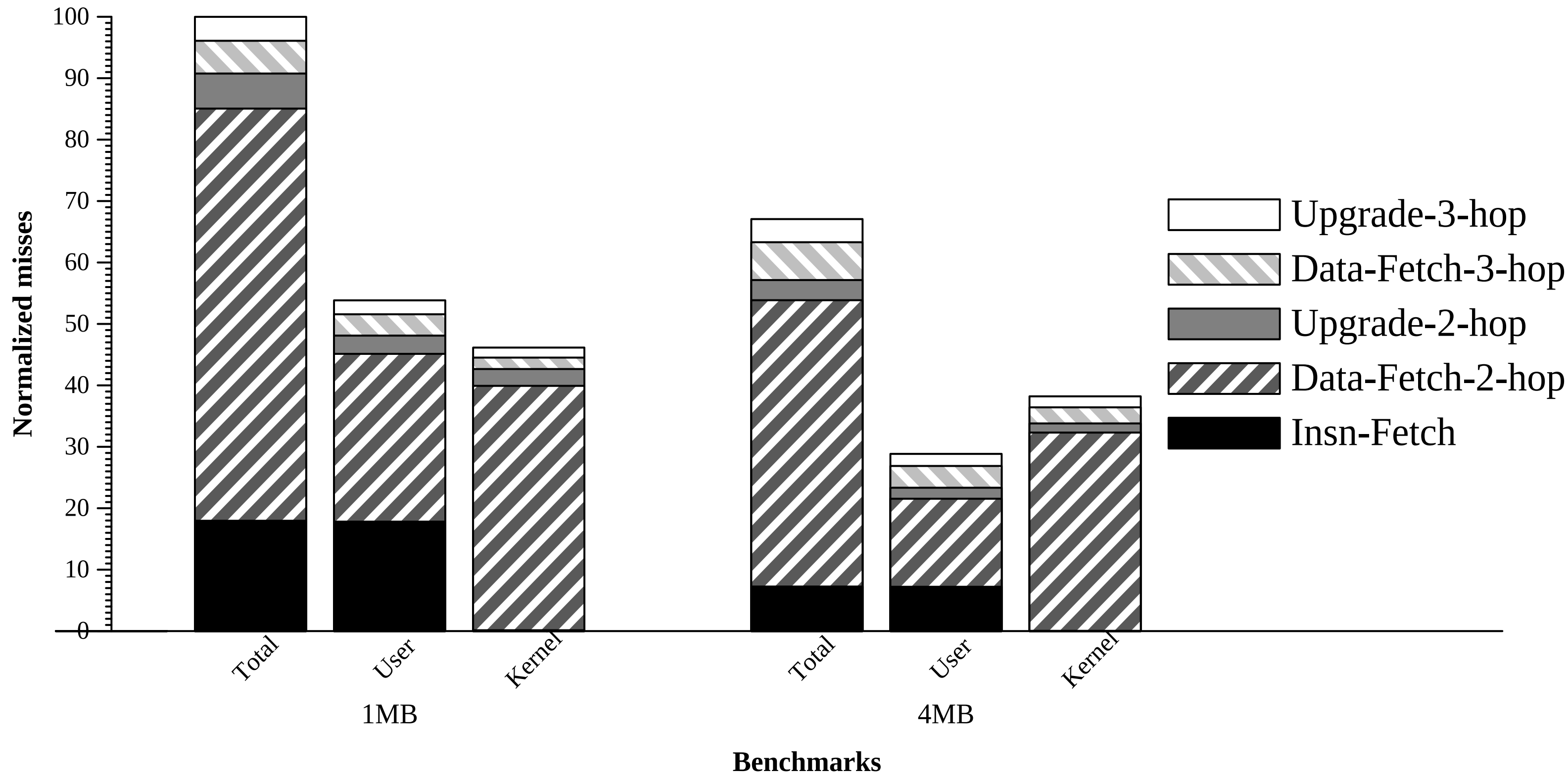
SimOS: Data Cache Miss Rate (2.5 B inst.), per memory access, 128-byte block, 4-way

Size	Load	Store	Total
512 KB	0.25 %	0.22 %	0.24 %
1 MB	0.19 %	0.19 %	0.19 %
2 MB	0.14 %	0.17 %	0.15 %
4 MB	0.11 %	0.15 %	0.13 %
16 MB	0.08 %	0.13 %	0.1 %

SimICS: Data Cache Miss Rate (300-450 M inst.), per instruction, 64-byte block, 4-way

Size	Total
1 MB	0.64 %
4 MB	0.43 %

Coherence (SimICS)



Future work

- More **workload tuning**
  - We found proper indexing important
  - Exploit **lazy consistency**
  - **Database tuning**
  - More efficient Java servlet/JDBC code
  - Better JVM/JIT
- SSL Implementation
- Larger database size
- **Multi-tier measurements**
- **Detailed architectural characterization**

Summary

- **A difficult and complex task**
  - **Most of the TPC-W specification implemented**
  - Setup of TPC-W on **two different full-system simulators**
- Using Java enables **study of two ISAs and OSs with no changes to the application**
- Performance tuning and optimization to be completed
- Preliminary characterization of this workload
- *Most complex workload setup under full-system simulation?*
- Look for updates and a **future source code release** at

http://www.ece.wisc.edu/~mikko/tpcw.html

Remote browser emulator (RBE)

- **Emulates web users** interacting through browsers
  - RBE manages a collection of Emulated Browsers (EBs)
  - **Each EB represents a single user**
- **Non-deterministic walk** over web pages
  - Send HTTP request
  - Parse HTTP response for images and other URLs
  - Wait for think time
  - Repeat
- Collects statistics required by TPC-W (**Matlab file**)
  - Throughput over time (**WIPS**)
  - Web interaction response times (**WIRT**)
  - Transaction mix

DB2 table sizes

13 indexes were added to speedup the queries

Table Name	Number of Rows	Maximum Bytes	Maximum Total Bytes
address	288,000	146	42,048,000
author	2,500	568	1,420,000
cc_xacts	129,600	89	11,534,400
item	10,000	869	8,690,000
country	92	70	6,440
customer	144,000	695	100,080,000
orderline	388,800	116	45,100,800
orders	129,600	81	10,497,600
Total	1,092,592	N/A	219,377,240