Evaluating the Effect of Microreboots on End Users

George Candea + Armando Fox

with help from
Ben Ling and Wes Weimer

January 12, 2004

Review

- Microreboots
  - restart fine-grained components "with a clean slate"
  - only take a fraction of the time needed for full system reboot
- They provide
  - a simple recovery technique for Internet services
  - which can be supported entirely in middleware and
  - requires no changes to apps or a priori knowledge of app semantics
- Low cost → use microreboots aggressively even when their necessity is less than certain
  - reduces recovery time
  - reduces time spent detecting/diagnosing failures

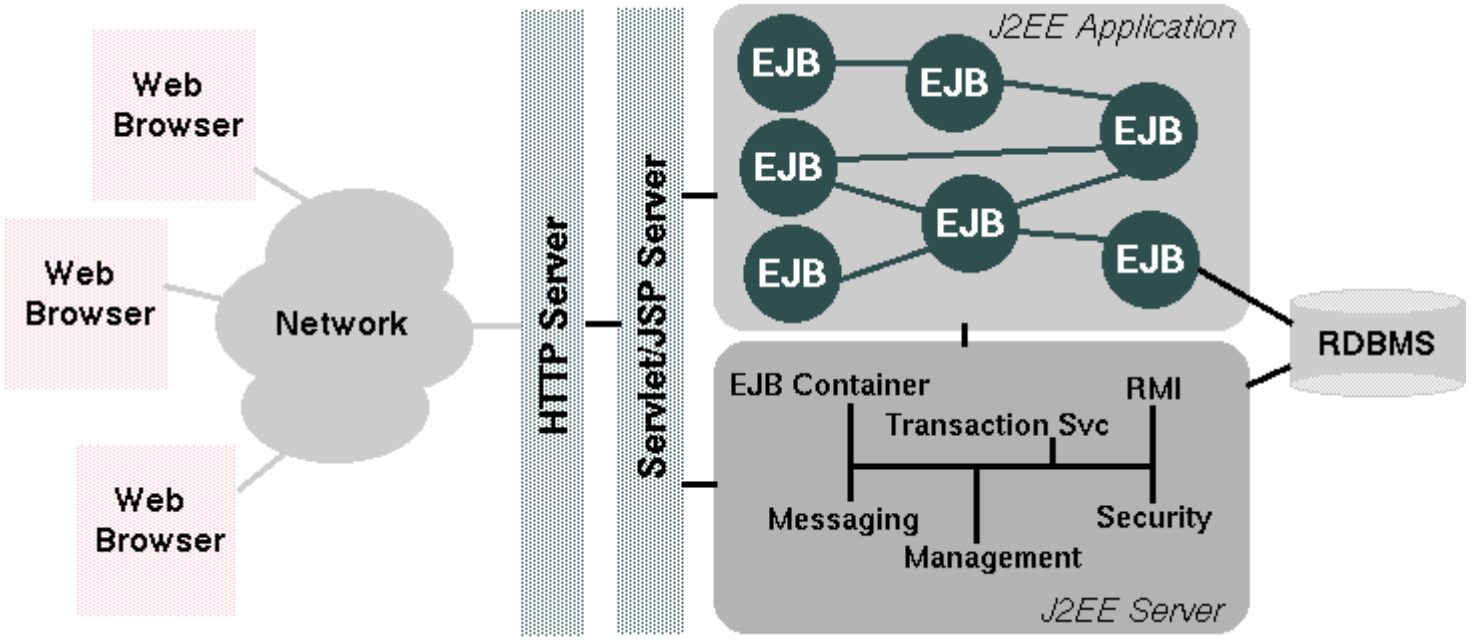
Goals and Preview

- Determine how infra microreboots impact end users
- Microreboots are less disruptive in terms of downtime and lost work (compared to full reboots)
- Reduced (compared to a server process restart)
  - number of failed user requests by 65%
  - perceived downtime by 78%

Outline

1. Measuring end user experience in interactive services
2. Session-weighted operation throughput
3. End user effects of microrebooting

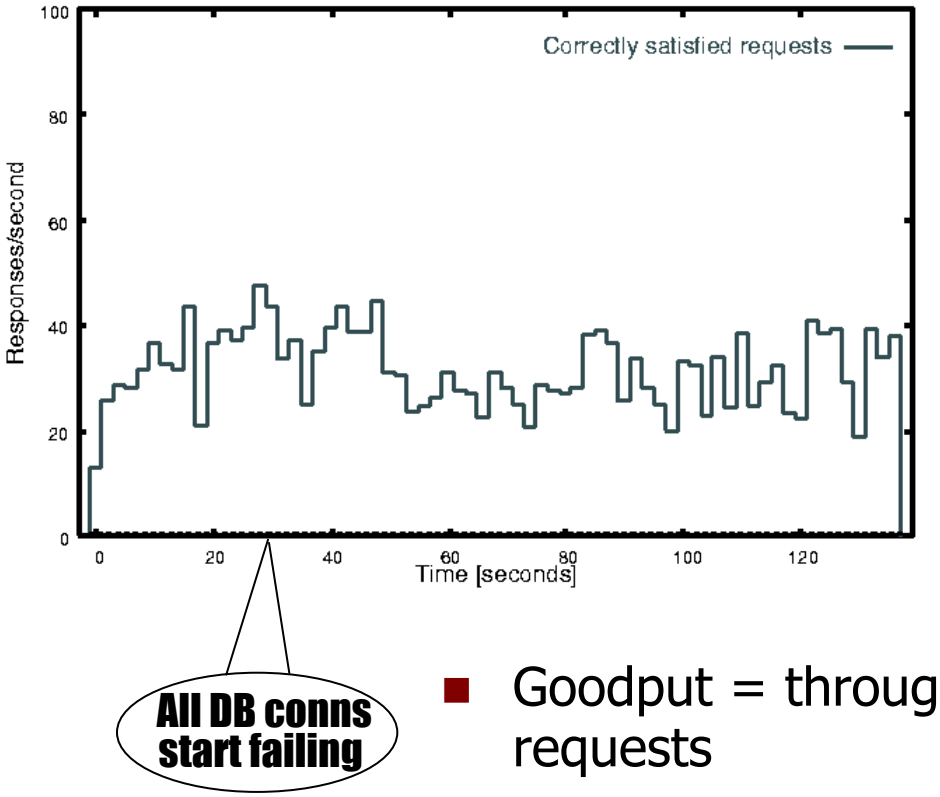
Three-Tiered Architecture



Simulating End Users

- RUBiS = open-source web-based auction application, modeled on eBay
- client states correspond to various RUBiS operations, such as *Register*, *PutBid*, etc. (27 in total)
  - non-app-specific states: user hits back button and user spontaneously decides to end session
- client workload described using a state transition table T
  - rows/columns = client's possible states
  - T(s1,s2) = probability of a client clicking from s1 to s2
- Emulator uses table T to automatically navigate the web site
  - when in state s1, randomly choose the next state based on T(s) with the requested probability
  - then generates corresponding URL and "clicks" on it
- T also has column for wait time inbetween clicking from a state to the next
  - we set this to zero → initiate next HTTP request as soon as the current request completes (unlike a real user)
- Responses classified as incorrect:
  - network-level error (cannot connect to server, etc.)
  - HTTP 4xx or 5xx error code
  - HTML page containing particular keywords ("error", "failed"', "exception")
- Use correct responses to compute goodput

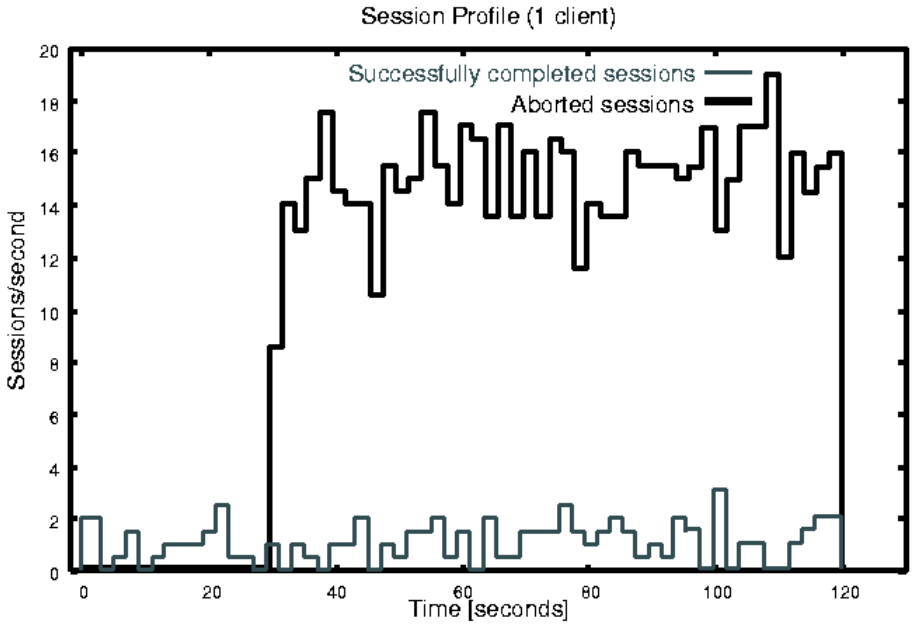
The Goodput Anomaly



- Goodput = throughput of successfully served requests
- Is the service available ? Is it delivering useful service?
- Can we increase goodput by failing components ? (see total throughput...)

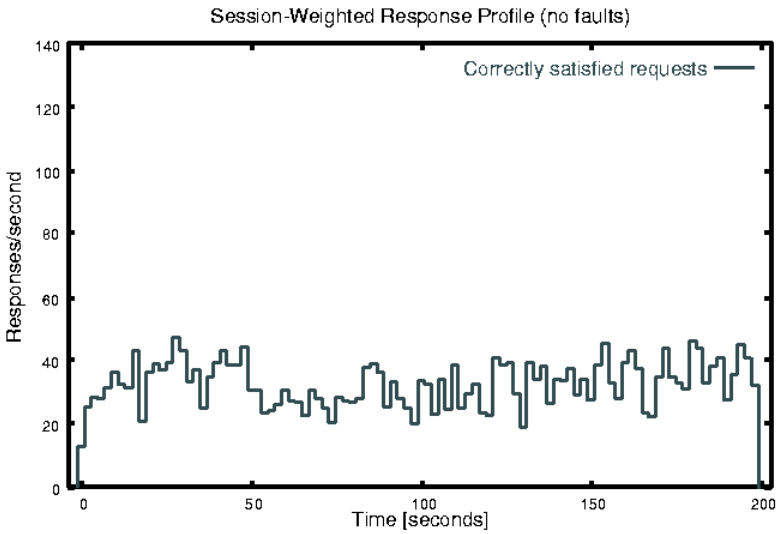
Sessions

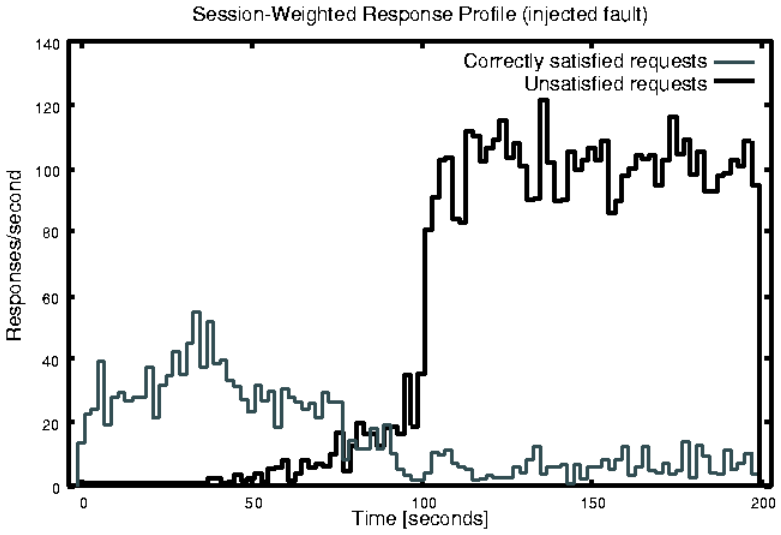
- Typical interaction of a client with the web site:
  - client goes to the homepage
  - browses around for a while performing different site actions (searching, etc.)
    - accumulates session state
  - decides to do something that touches the persistent-state database (e.g., place a bid, update his/her profile, etc.)
- assume interactions preceding the persistent-state update are just precursors to crowning moment
- failed op during session → retry or logout+login (typically at homepage)
- session = sequence of URLs bracketed by homepage and either abandonment of site or return to homepage (new session)
- definition relies on general user behavior, not app semantics
- Goodput anomaly explanation
  - → discard requests in failed sessions



The G_{wop} Metric

- Session-weighted goodput (G_{wop})
  - weighs each session by the number of operations in it
  - measures standard throughput of successful/failed requests respectively, but whenever a session fails, all ops in that session are counted as failed
- Conservative metric for recovery measurements
- G_{wop} captures the fact that when a long session succeeds, the user got a lot more done than when a short session succeeds





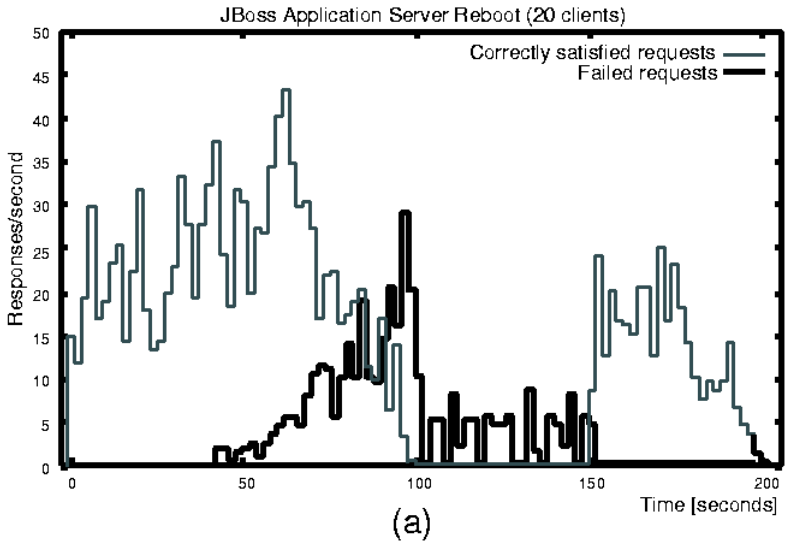
Outline

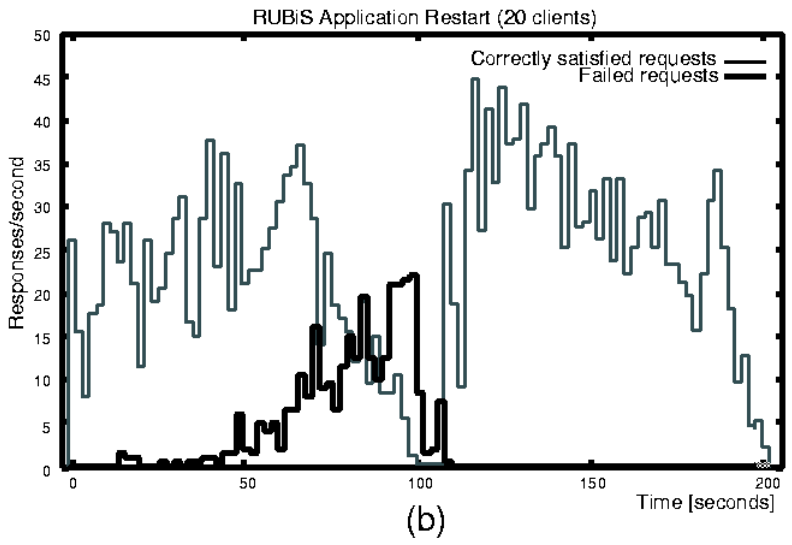
1. Measuring end user experience in interactive services
2. Session-weighted operation throughput
3. End user effects of microrebooting

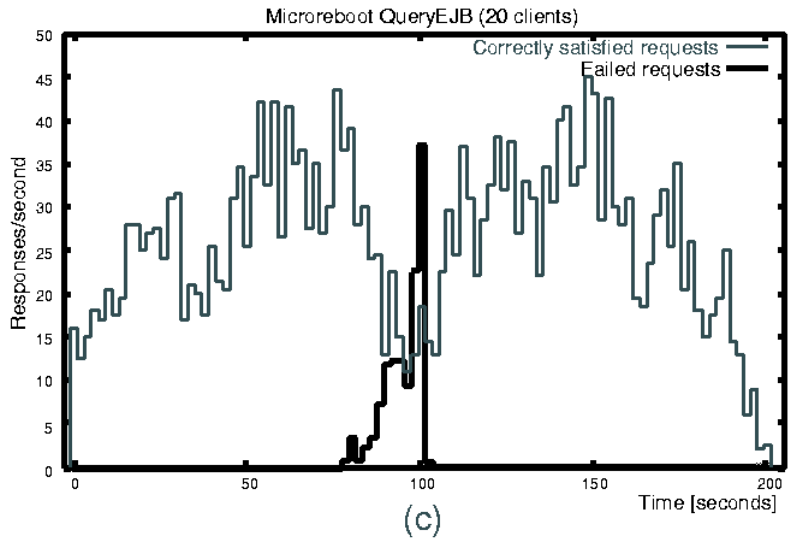
Effect of Microreboots on End Users

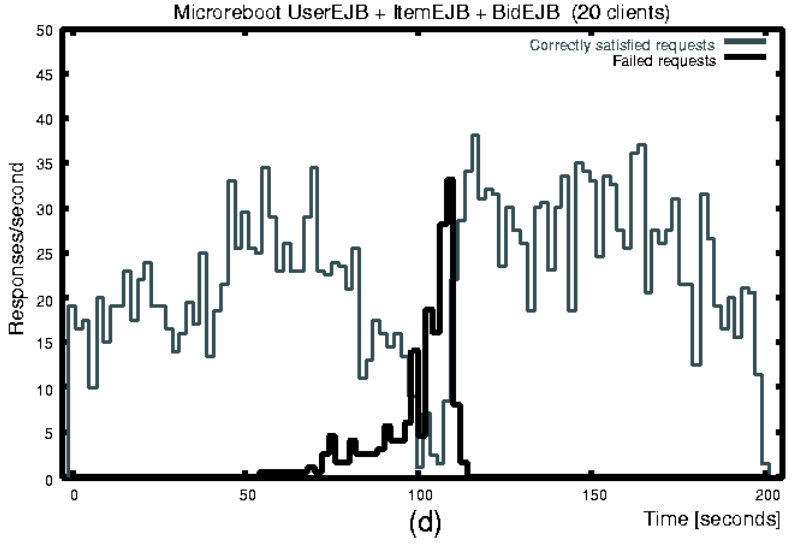
- Analogous to rebooting, a microreboot is a logical restart of an application component that may be finer-grained than a process
- Basic microrebootable component of J2EE applications is the EJB
- Leverage existing JBoss mechanism for cleanly "shutting down" an EJB
- Isolate microreboot from fault detection → initiate reboot w/out injecting faults
  - i.e., app server instantaneously detects fault and initiates recovery
- chosen workload covers all possible RUBiS operations
  - runs lasting 1 minute or longer routinely exercised all components
  - workload mix: 85% read ops and 15% DB write ops

G_{wop} for Reboot-Based Recovery







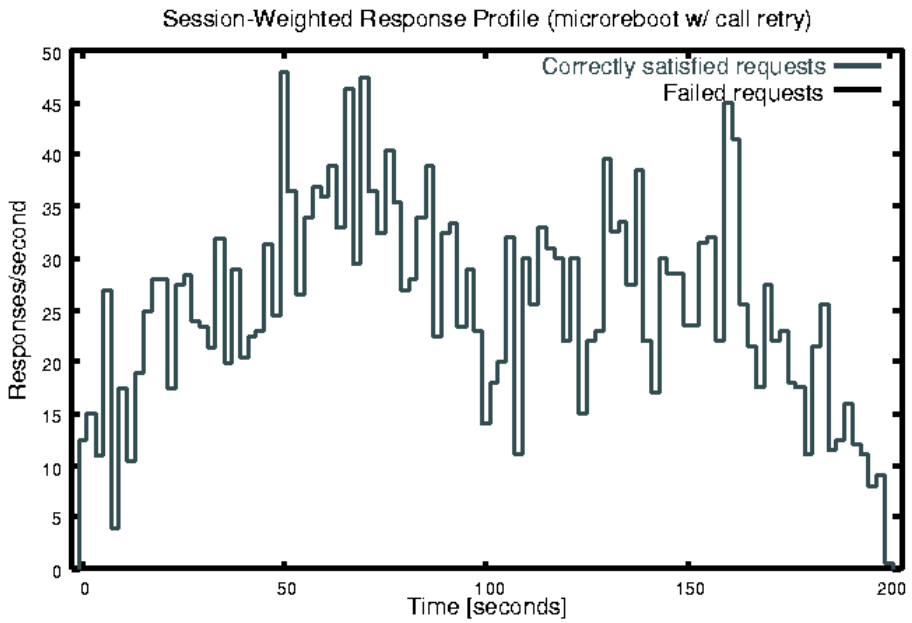


Summary of Results

Recovery Technique	Failed Reqs	Downtime [sec]	Improvement (served reqs)	Improvement (downtime)
JBoss Restart	713	108	--	--
RUBiS Restart	615	94	14%	13%
1-EJB microreboot	251	24	65%	78%
3-EJB microreboot	345	29	52%	73%

Improving End User Experience w/ Transparent Retry

- Expose EJB microreboot through a RetryLater(t) exception
- Modified EJB container catches RetryLater(t) exceptions and retries transparently
- Call succeeds after predetermined number of retries → original bean code sees successful call
  Call fails → container throws exception to the caller
- Masks microreboots (and transient failures) from callers
- Idempotency: invocations are atomic (call either goes through or RetryLater is thrown)
  - preserves JBoss's regular call semantics
- In-progress calls fail (upon microreboot) the same way as if the EJB crashed, had a bug, etc.



Feedback

http://crash.stanford.edu

- Better name for G_{wop} ?

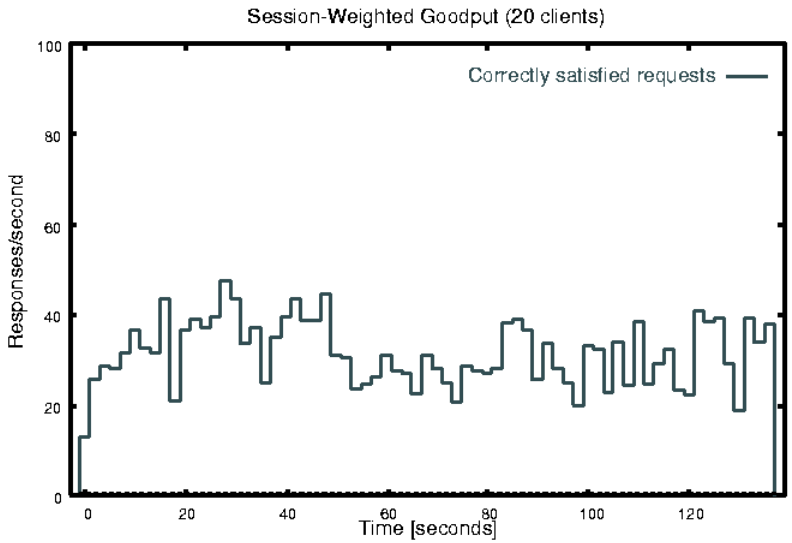
Details on Microreboots

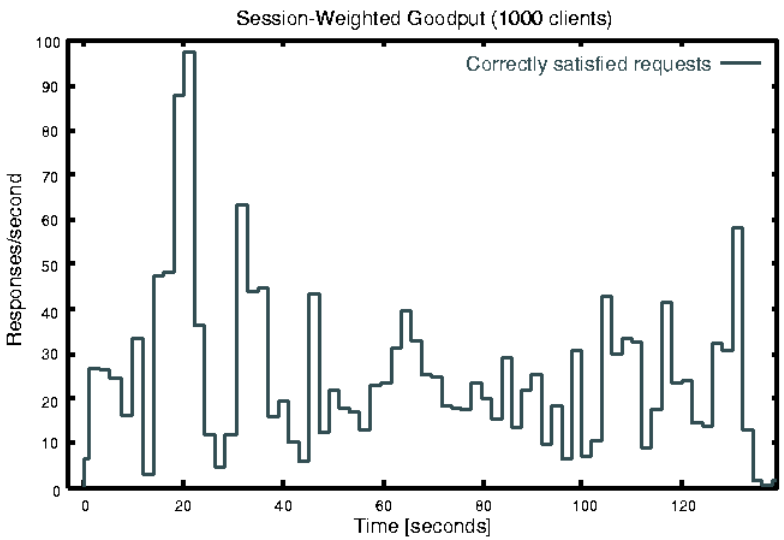
- Application-generic recovery based on checkpointing applies to relatively few existing applications
- Part of the appeal of rebooting as a recovery tech is that it *discards* corrupted transient state that might itself be the cause of the failure or whose cleanup may be necessary in order for recovery to succeed
- Replace recovery with rebooting (logically equivalent to restarting from a checkpoint that is the start state of the component) is likely to work
- To ensure that it is safe, we need three environmental conditions:
  - Clear boundary around what is being rebooted
  - Loose coupling
  - Preserving state and consistency
- Analogous to rebooting, a microreboot is a logical restart of an application component that may be finer-grained than a process, but the same requirements apply
- Our basic microrebootable component of J2EE applications is the EJB

RUBiS Details

- RUBiS = open-source web-based auction application, modeled on eBay
- offers selling, browsing and bidding
- three kinds of users: visitor, buyer, and seller, with buyer and seller sessions requiring login
- buyer can bid on items and consult a summary of their current bids, rating and comments left by other users
- Seller sessions require a fee before a user is allowed to put up an item for sale
- Seller can specify a reserve (minimum) price for an item
- RUBiS contains 582 Java files and about 26K lines of code
- Uses MySQL and stores 7 tables
- In default configuration, RUBiS has 33,000 items for sale, distributed among eBay's 40 categories and 62 regions. There is an average of 10 bids per item, or 330,000 entries in the bids table. The users table has 1 million entries.

Why choose 20 clients ?



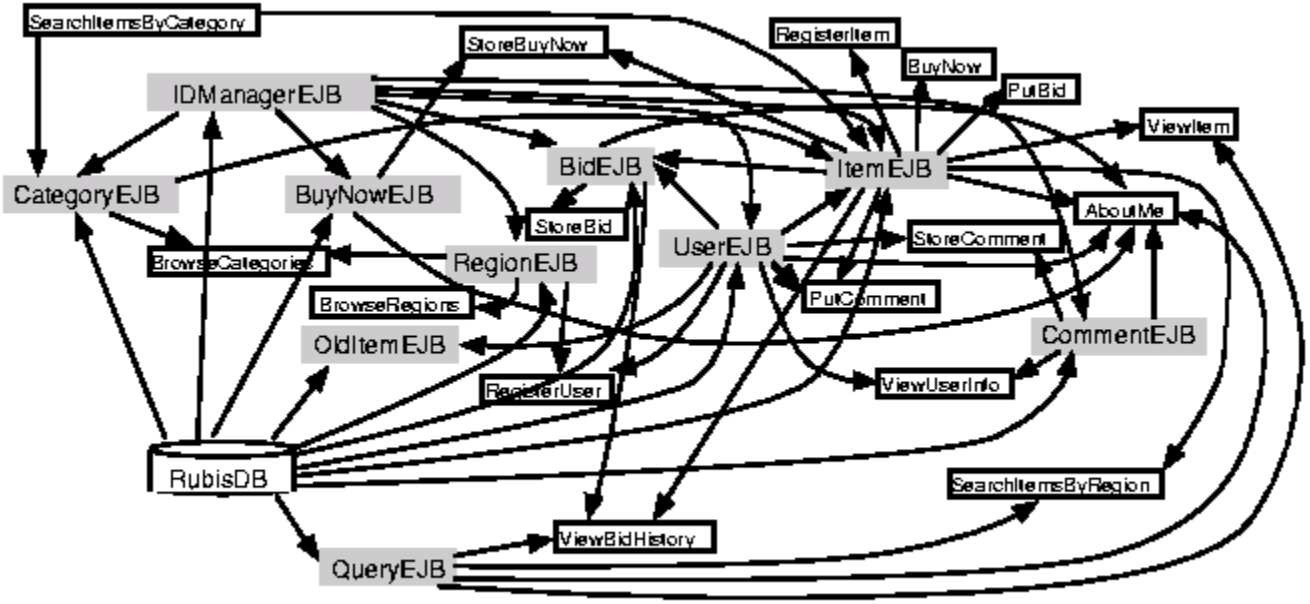


- 20 concurrent clients with no think time inbetween successive requests
- human user typically spends in excess of 2 seconds between clicks, we believe the load placed by one of our simulated clients is equivalent to that of 100 or more real clients
- did not want to introduce artificial think time (the way is done, for instance, in the TPC-W benchmark) because having think time would add one more variable to the experiment and would not offer any useful insight for microreboot experiments
- 20 concurrent rapid clients is the threshold beyond which thrashing and other side effects would reduce throughput

Experimental Platform

- mimic what would be typical of a small Internet service
- Linux RedHat 9.0
- JBoss and the Web tier run on an AMD Athlon XP 2600+ PC with 1.5 GB RAM
- Database (MySQL Max 3.23) on another identical node
- Sun HotSpot JVM 1.4.1. and allocate it 1 GB of RAM
- Client emulator on a dual P-III (2x866 MHz) with 1 GB RAM
- All machines interconnected by 100 Mbps Ethernet switch

RUBiS f-map



- description of the failure dependencies between RUBiS's components using automated fault-propagation inference (AFPI)
- majority of such dependencies in RUBiS are between the stateless servlets and EJBs
- AFPI information is collected during a completely automated fault-injection campaign that requires no a priori knowledge of the applications' structure or semantics
- For RUBiS, a simple recovery policy could be based on the fact that there is a known mapping from the URL being accessed to the action being taken (and hence the EJBs being touched)