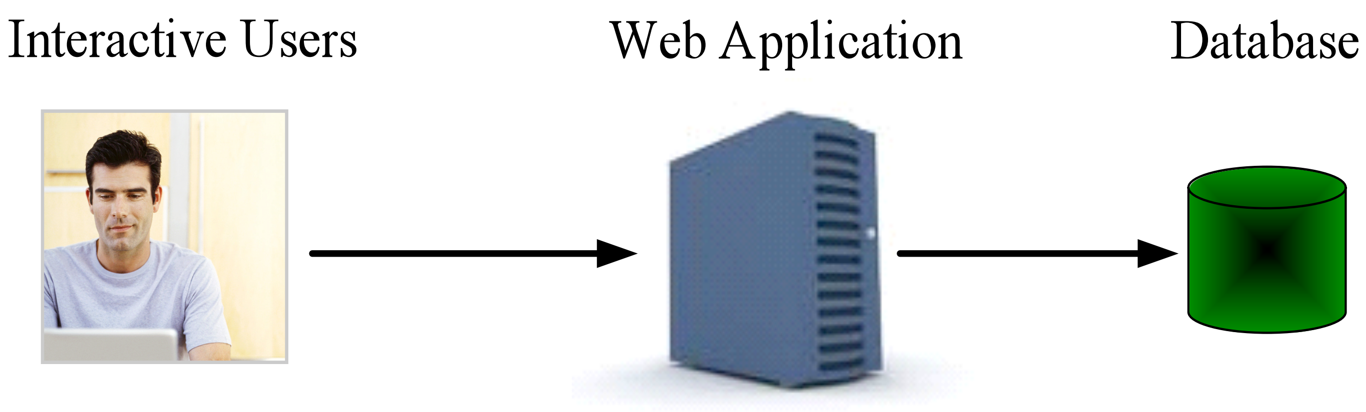
Black-Box Performance Control for High-Volume Non-Interactive Systems

Chunqiang (CQ) Tang IBM T.J. Watson Research Center
Sunjit Tara IBM Software Group, Tivoli
Rong N. Chang IBM T.J. Watson Research Center
Chun Zhang IBM T.J. Watson Research Center

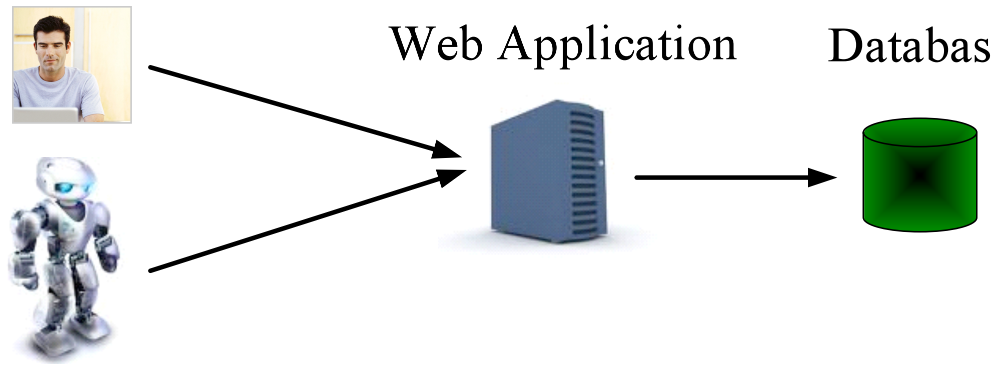
UENIX'09, June 19, 2009

Response Time Driven Performance Control for Interactive Web Applications



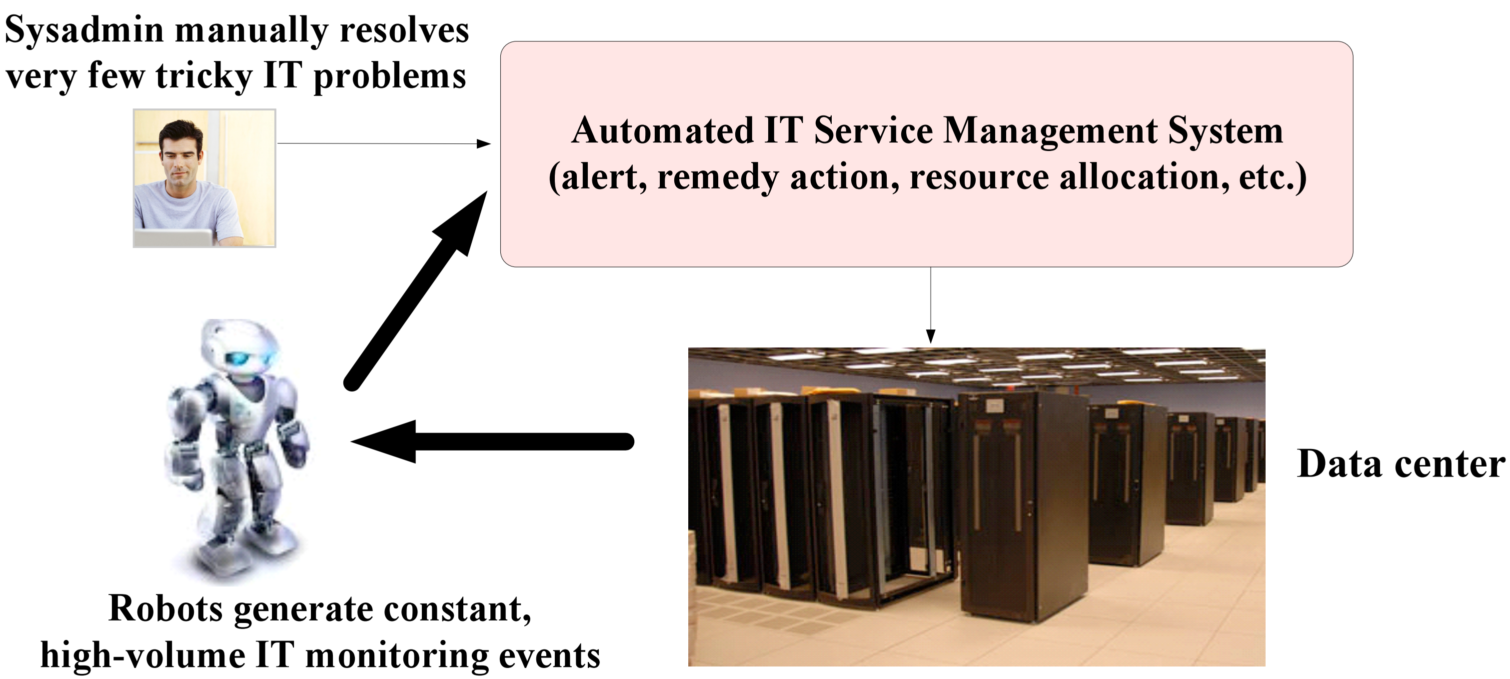
- Interactive users are sensitive to sub-second response time
- Naturally, performance control is driven by response time
  - E.g, stop admitting new requests if response time exceeds a threshold
  - Well studied area: admission control, service differentiation, etc.

But there are Robots that Impact Perf Control



- Many Web services also provide APIs to explicitly work with robots
  - Twitter API Traffic was 10x of its Web traffic
- Some applications work with interactive users during daytime, and then are driven by robot tools at nights to perform heavy-duty analytics
- How robots impact performance control
  - They often have tons of work to do and hence are throughput centric
  - They may not require sub-second response time, e.g., crawler and analytics

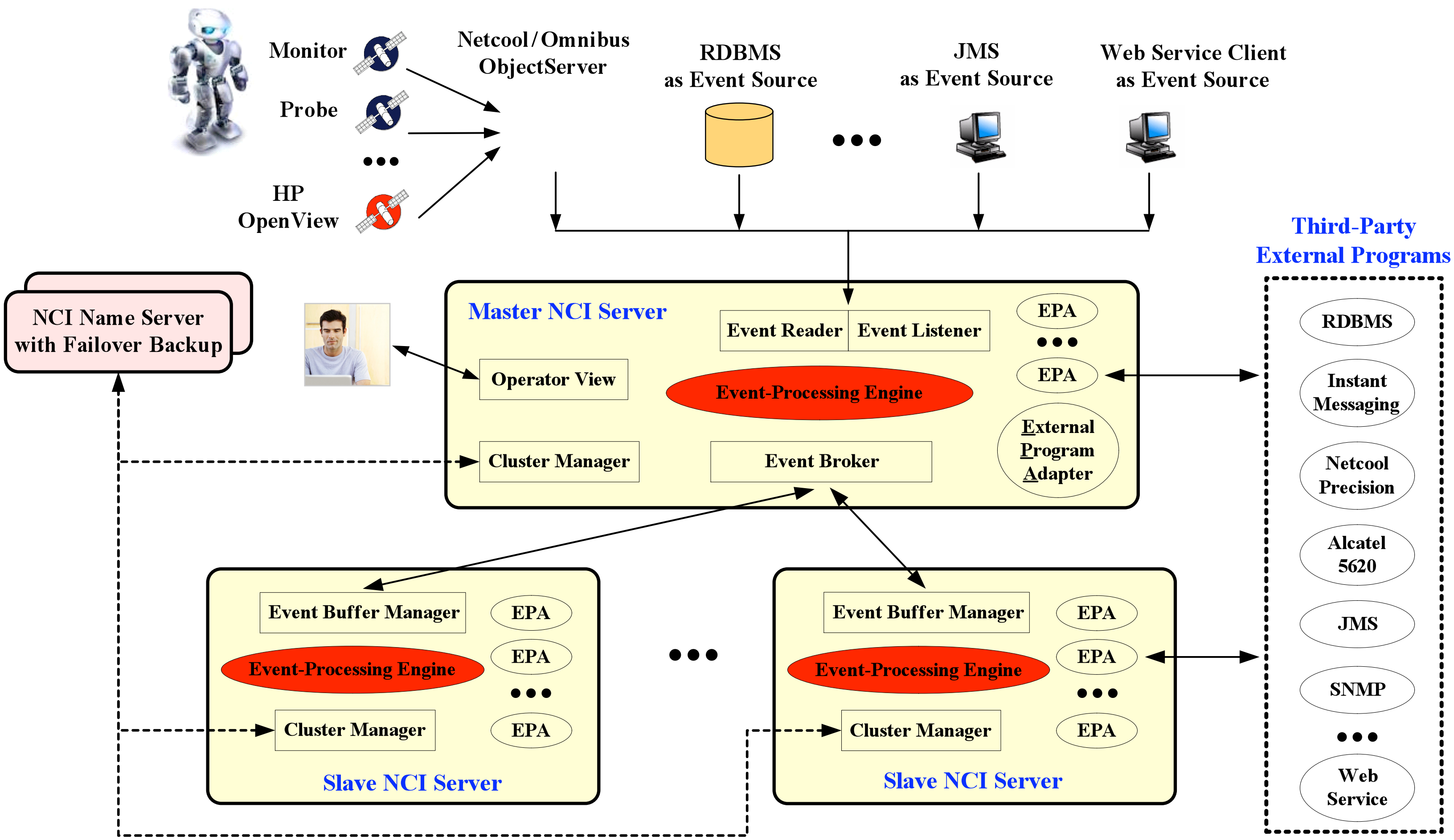
IT Monitoring and Mgmt: a World where Robots Rule



- Before an IT service mgmt system (ITSM) can manage a data center, it must manage itself well
  - Withstand event flash crowd triggered by, e.g., router failure
  - Achieve high event-processing throughput by driving up resource utilization
  - Avoid resource saturation as sysadmins may want to do manual investigation

Simplified View of IBM Tivoli Netcool/Impact

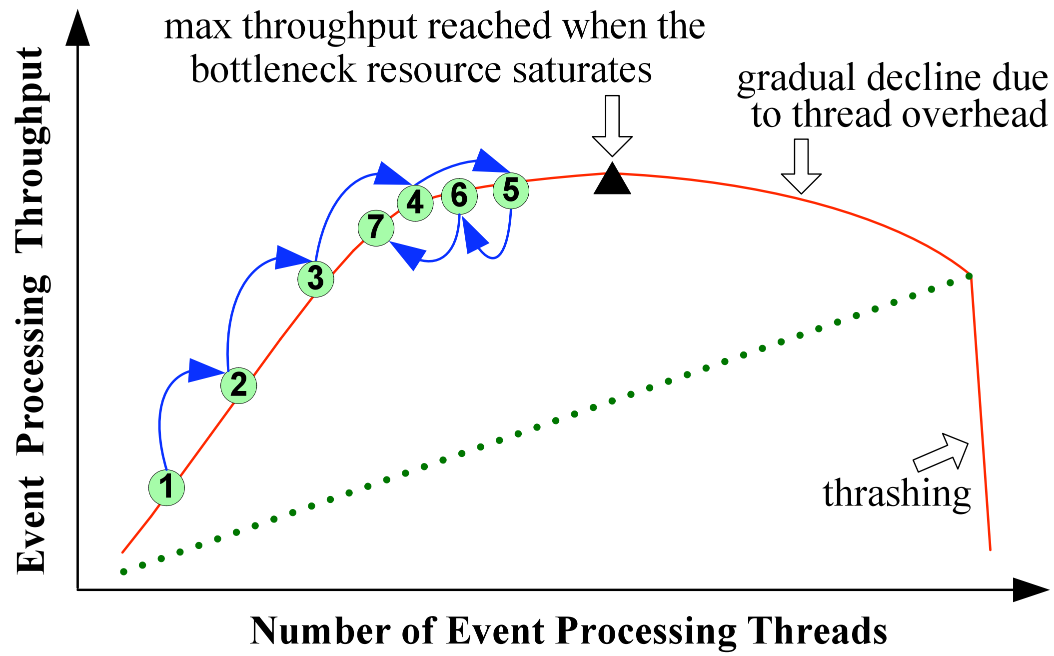
- It provides a reusable framework for integrating all kinds of siloed monitoring and mgmt tools
- It is built atop a J2EE engine but cannot use response-time driven performance control



Why Perf Control is Difficult in Netcool/Impact

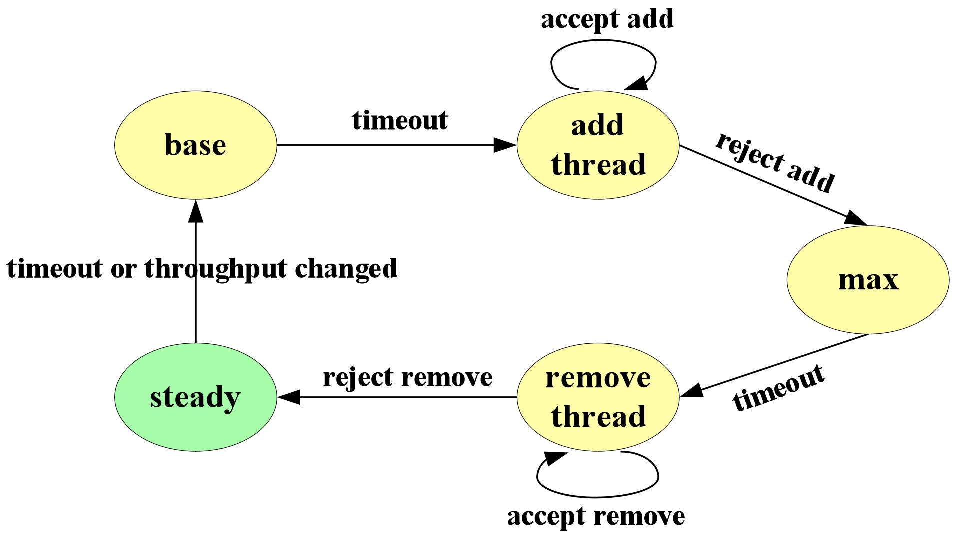
- Work with third-party software provided by many vendors
- We cannot greedily maximize performance without considering congestion
- Bottleneck can be anything anywhere: CPU, disk, memory, network, etc.
- Bottleneck depends on how users write their code atop Netcool/Impact
- Not a simple static topology like web->app->DB
- No simple perf indicator like packet loss or response time violation

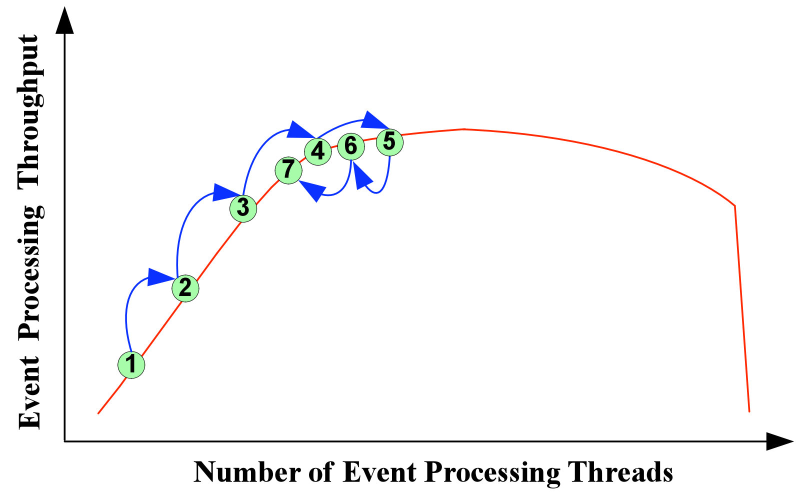
Black-Box Approach: Throughput-guided Concurrency Control (TCC)



- Why not simply use TCP to maximize throughput
  - We deal with general distributed systems rather than just network
  - No packet loss as performance indicator
  - Unlike router, a general server’s service time is not a constant

Simplified State-Transition Diagram for Thread Tuning





- base state: reduce threads by w%
- add-thread state: repeatedly add threads so long as every p% increase in threads improves throughput by q% or more
- remove-thread state: repeatedly remove threads by r% each time so long as throughput does not decrease significantly

Conditions for Friendly Resource Sharing

- Repeatedly add threads so long as every p% increase in threads improves throughput by q% or more

$$q > \frac{p(p+1)}{p+2}$$

e.g., double threads (p=100%) and then see thruput increases by q=1%. This is no good.

- Reduce threads by w% at the beginning of exploration

$$w \geq 1 - (\frac{p}{q} - 1)^2$$

The base state must be sufficiently low so that it will end up with less threads if resource is saturated

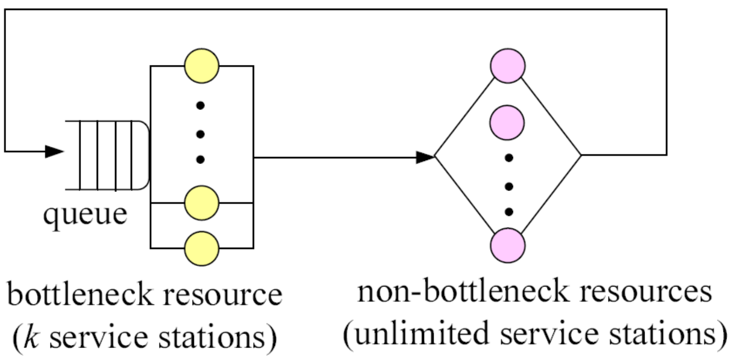
Conditions for Friendly Resource Sharing

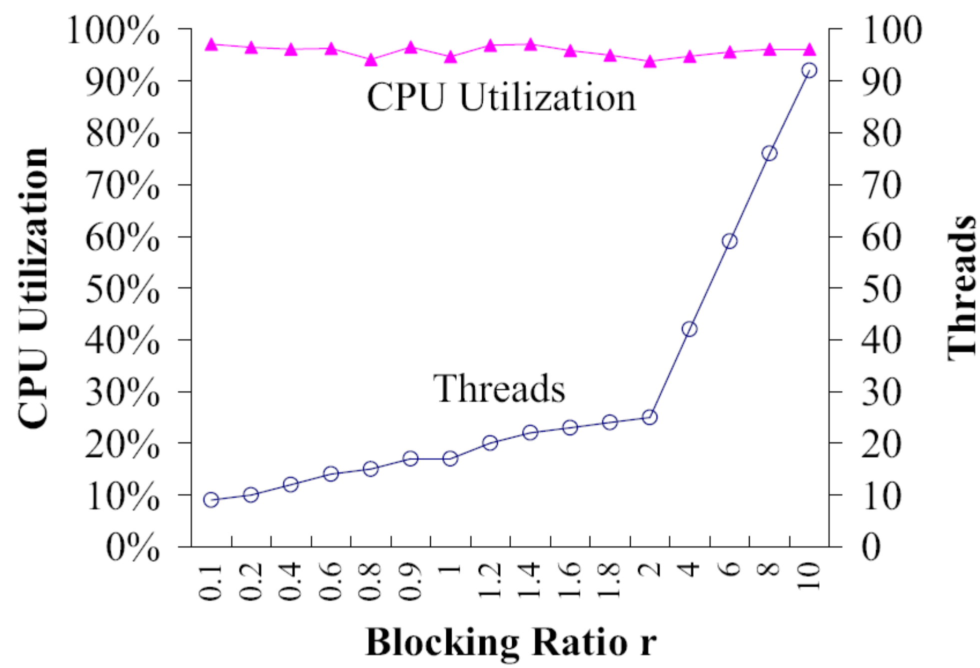
$$q \quad > \quad \frac{p(p+1)}{p+2} \qquad\qquad w \quad \geq \quad 1-(\frac{p}{q}-1)^2$$

- If there is an uncontrolled competing program, NCI shares 44–49% of the bottleneck resource
- Two instances of NCI share bottleneck resources in a friendly manner
- However, three or more instances of NCI need coordination from the master

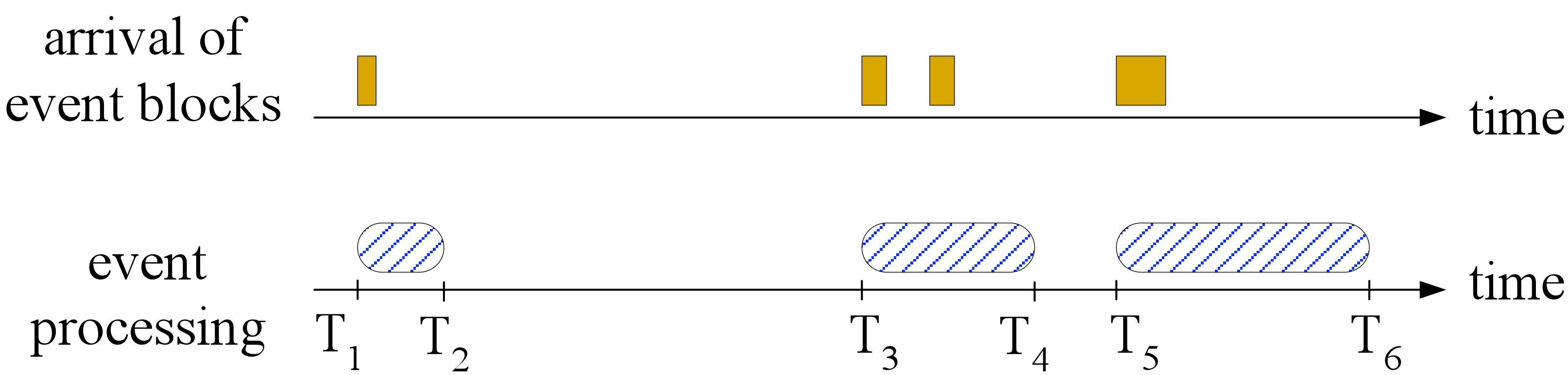
Drive up Resource Utilization to Achieve High Throughput

- TCC is friendly but also sufficiently aggressive to drive up resource utilization





Throughput Measurement 1: Exclude Idle Time from Throughput Calculation



$$\text{Throughput} = \frac{n}{T_6 - T_1}$$

$$\text{Throughput} = \frac{n}{(T_2 - T_1) + (T_4 - T_3) + (T_6 - T_5)}$$

Throughput Measurement 2: Minimize Measurement Samples

- Minimize the number of measurement samples while ensuring a high probability of making correct decisions

Problem formulation

Minimize $n = n_1 + n_2$

Subject to

$$\sigma_y^2 = \frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2} \leq \left\{ \frac{\max(H - \mu_y,\ \mu_y - L)}{Z_{1-\alpha}} \right\}^2 \quad (18)$$

$$n_1, n_2 > 0 \quad (19)$$

Solution

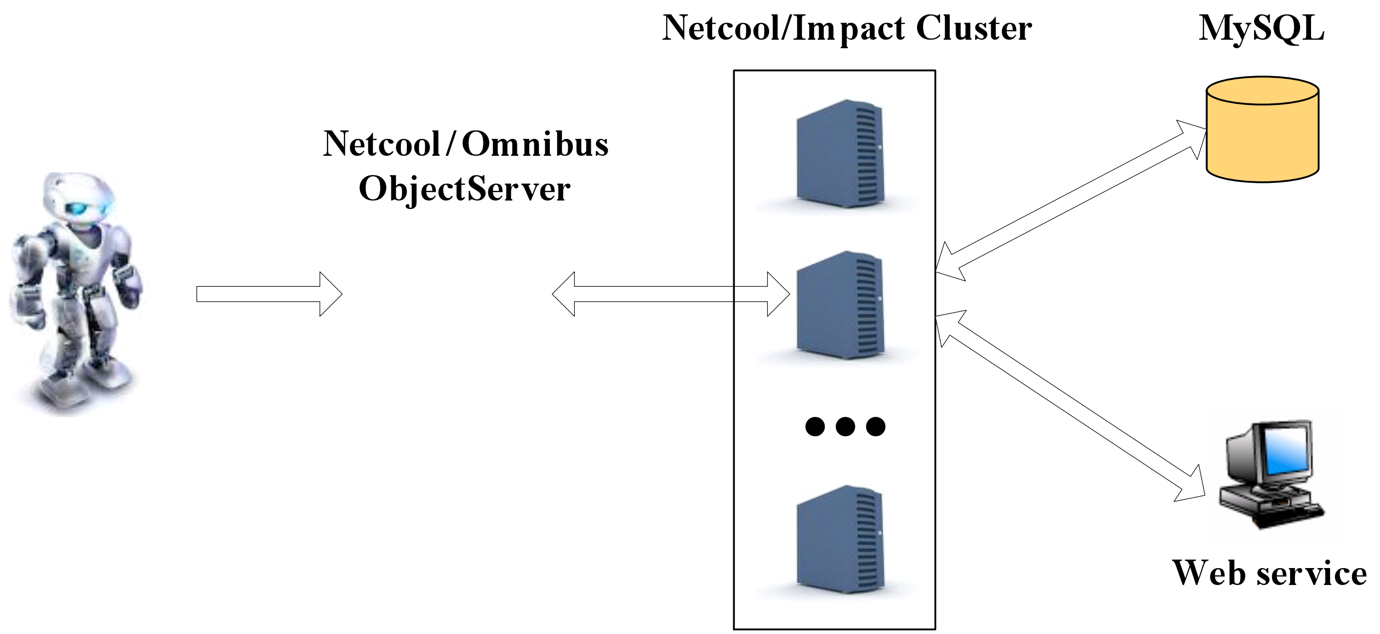
$$\widehat{n}_1 = \sigma_1(\sigma_1 + \sigma_2) \left\{ \frac{Z_{1-\alpha}}{\max(H - \mu_y,\ \mu_y - L)} \right\}^2$$

$$\widehat{n}_2 = \sigma_2(\sigma_1 + \sigma_2) \left\{ \frac{Z_{1-\alpha}}{\max(H - \mu_y,\ \mu_y - L)} \right\}^2$$

Throughput Measurement 3: Exclude Outliers from Throughput Calculation

- Extreme activities such as Java garbage collection introduce large variance
  - Sometimes GC can take as long as 20 seconds
- There are many known methods to handle outliers
- We found that simply dropping 1% of the largest samples works well
- This is simple but critical

Experimental Setup



- In some experiments, we introduce extra network delay
- In some experiments, we control service time of the Web service and Netcool/Impact user scripts

Scalability of NCI Cluster

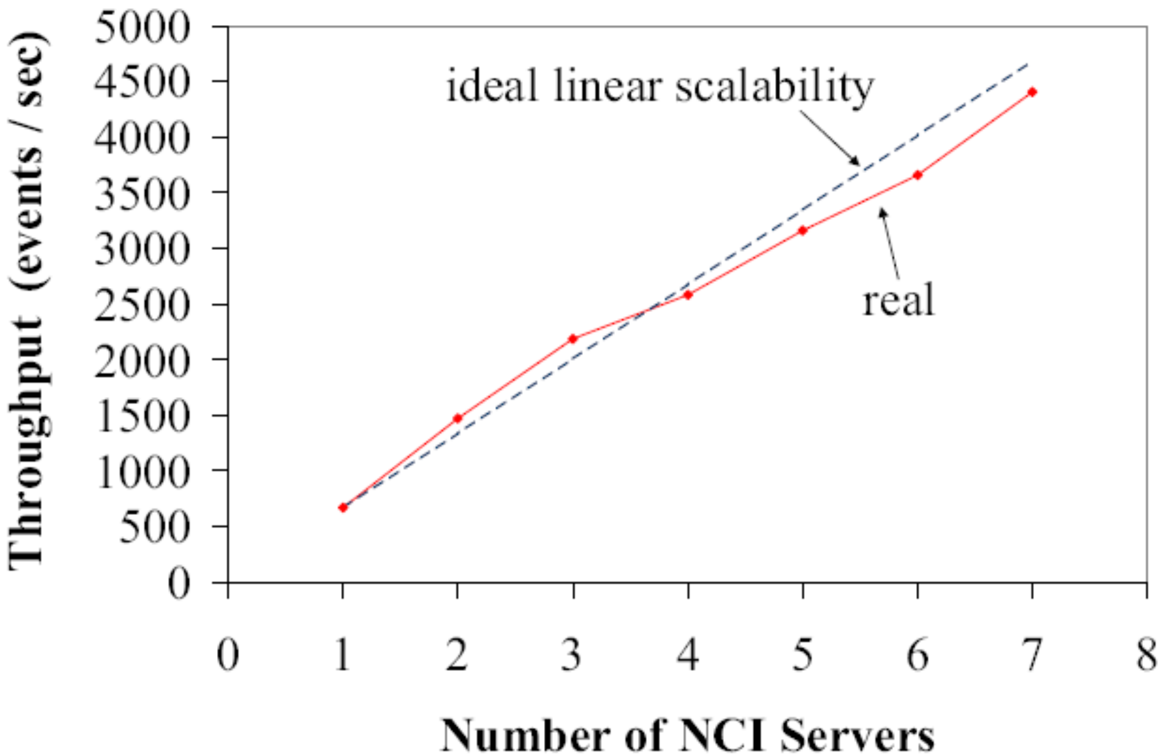
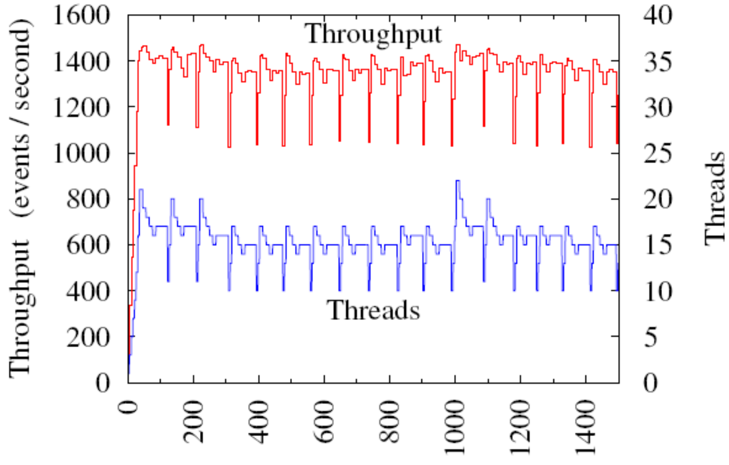
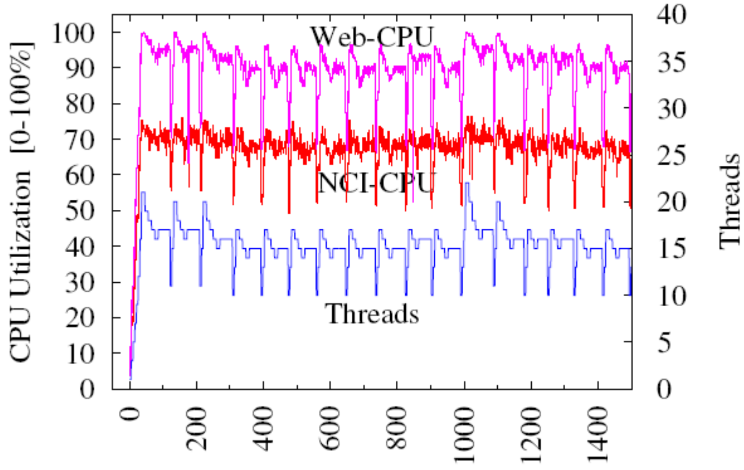


Figure 7: Scalability of NCI.

CPU as the Bottleneck Resource



(b) Throughput



(a) CPU Utilization

Recover from Memory Thrashing

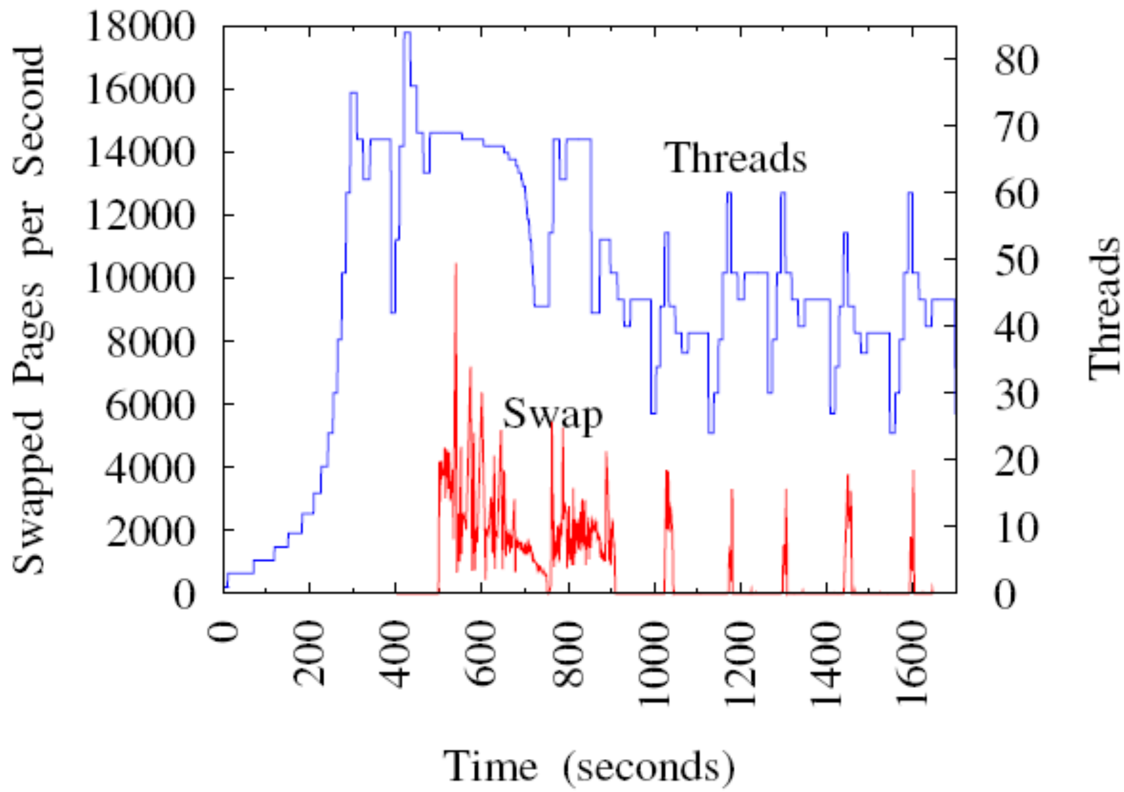


Figure 9: Memory bottleneck and memory thrashing.

Disk as the Bottleneck

Reducing threads actually improves disk performance

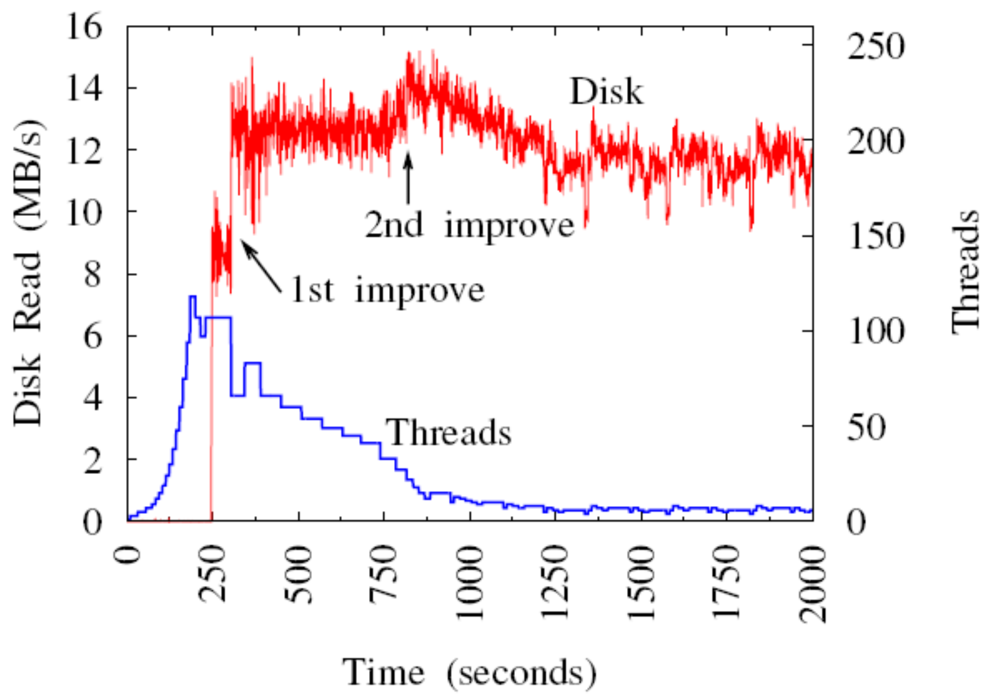


Figure 10: The Web machine's disk is the bottleneck. Removing threads actually improves disk throughput.

Work with an Uncontrolled Competing Program

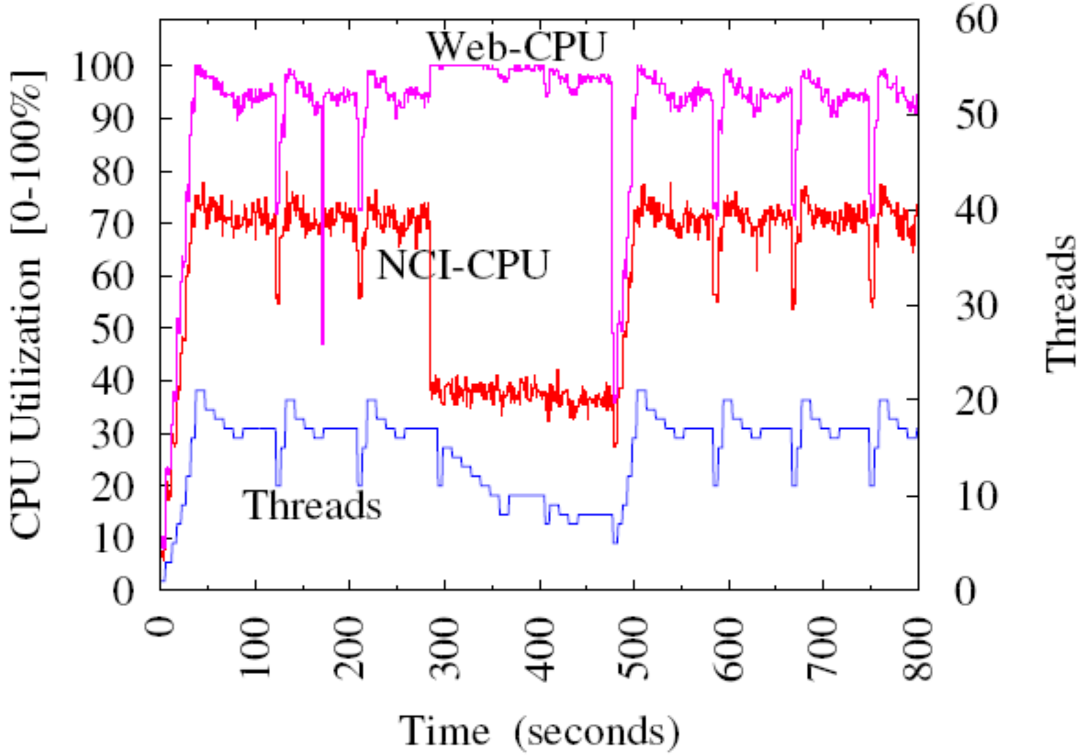


Figure 12: An external program competes for the bottleneck resource, which is the Web machine's CPU.

Related Work

- Greedy parameter search
  - Too greedy without considering resource contention
- TCP-style congestion control, e.g., TCP Vegas
  - Assume minimum RTT is the mean service time
  - In DB, min response time is the best-case cache hit service time. It cannot be used to estimate the congestion-free baseline throughput.
- Control theory
  - Not sufficiently black-box
  - Need to monitor resource utilization if applied to Netcool/Impact
- Queueing theory
  - Assume a known static topology and a known bottleneck

Future Work

- Is it possible to get “TCP-friendly” for general distributed systems?
  - Currently three or more instances of NCI need coordination in order to be friendly to each other
- Can we estimate the utilization of Google’s internal servers by observing changes in query response time?
  - This is possible for restricted queuing models
  - What’s the most general model for which this is still doable?

Take Home Message

- We need to revisit performance control for systems that handle workloads generated by software tools (robots)
  - Mixed human/robot worklaod (Twitter fits here)
  - Mostly robot workload (Netcool/Impact fits here)
  - Robot-only workload (Hardoop fits here)