Fundamentals of Software Performance

METHOD R™

Quick Reference Card

Glossary

coherency delay The duration that a task spends communicating and coordinating access to a shared resource, measured in time per task execution: *seconds per click*.

efficiency A measure of how much resource is consumed by an execution of a desired task. Efficiency is the inverse of how much of a task execution's total service time can be eliminated without increasing capacity, and without sacrificing required business function.

knee The utilization value for a resource at which throughput is maximized with minimal negative impact to response times. The knee value for a given number of service channels is less than or equal to the value shown here:

# Service channels	1	2	4	8	16	32	64	128
Knee utilization	50%	57%	66%	74%	81%	86%	89%	92%

load Competition for a resource induced by concurrent task executions, measured as utilization.

performance The speed of task execution, measured either as throughput or response time.

profile A tabular decomposition of response time, typically listed in descending order of component response time contribution.

queueing delay The duration that a task spends enqueued at a given resource, awaiting its opportunity to consume that resource, measured in time per task execution: *seconds per click*.

response time The execution duration of a task, measured in time per task: *seconds per click*. Response time is the sum of queueing delay and service time.

sequence diagram A diagram specified in the Unified Modeling Language (UML), used to show the interactions between objects in the sequential order that those interactions occur. A sequence diagram is useful in visualizing response time.

service channel A sub-resource that shares a single queue with other such sub-resources, such as a toll booth in a toll plaza or a CPU in an SMP computer.

service time The duration that a task spends consuming a given resource, measured in time per task execution: *seconds per click*.

task A business-oriented unit of work. Tasks can nest: *print invoices* is a task; *print one invoice*—a sub-task—is also a task.

throughput The count of task executions that complete within a specified time interval: *clicks per second*.

utilization Resource usage divided by resource capacity for a specified time interval; a quantitative measure of load.

When you think of performance, think of Method R:

- Software tools
- Project reviews
- Custom development
- Troubleshooting

© 2009–2010 Method R Corporation ◇ *http://method-r.com* ◇ *info@method-r.com* ◇ +1 817 251 0938

v2010.01.07

Fundamentals of Software Performance

METHOD R™

Quick Reference Card

Principles

1. A *sequence diagram* is a helpful tool for conceptualizing response time, but for tasks that execute thousands of calls (even dozens), the *profile* is more helpful.
2. Throughput is not the reciprocal of response time. Adding load to create higher throughput *changes* response time by adding *queueing delay* and possibly *coherency delay* as well.
3. The difference between a task's response time on a resource under light load and its response time under heavy load is its queueing delay.
4. A task's queueing duration for a given resource depends on that resource's number of *service channels* and its *load*:
   a. Adding service channels reduces queueing delay for a given load. But the benefits diminish as you add service channels.
   b. Reducing load reduces queueing delay for a given architecture. There are no scalability limits to load reduction.
5. The *knee* is the utilization value that defines the threshold between light load and heavy load upon a resource.
   a. On a heavily loaded resource, response times degrade exponentially as load increases.
   b. On a heavily loaded resource, response times improve exponentially as load decreases.
6. For systems with randomly timed service requests, allowing sustained resource loads in excess of the knee value results in severely degraded response times and throughputs that will fluctuate severely with microscopic changes in load.
   a. Hence, it is vital to manage load so that it will not exceed the knee value.
7. For a given set of computer resources, you cannot improve performance by making code *run faster*; you can improve performance *only* by eliminating unnecessary instructions in your code, or in the code competing against your code for resources.
8. Improving code path efficiency reduces service time *and* load, both of which improve response time, and both of which improve performance exponentially on heavily loaded resources.
9. You cannot optimize throughput for an inefficient task. To analyze a task's efficiency, you must analyze its response time. Thus, to optimize throughput, you must *first* analyze response time.
10. You will not reliably guess where a program spends its time, and it is inefficient to try to optimize everything you can think of. *Profiling* helps you write faster code, faster.

When you think of performance, think of Method R:

- Software tools
- Project reviews
- Custom development
- Troubleshooting

© 2009–2010 Method R Corporation ◇ *http://method-r.com* ◇ *info@method-r.com* ◇ +1 817 251 0938

v2010.01.07