Consequences of Resource Architectures in Distributed Systems

Murray Woodside, Carleton University, Ottawa, Canada
http://www.sce.carleton.ca/rads

Background

- **Performance Engineering of Real-time Software**
- **Workshop on Software and Performance**

Outline

- **Resources and architecture**
- **How resource architecture emerges from software design**
- **Serious consequences: logical bottlenecks**
- **Thoughts on logical resources**
- **Characterizing architectures**

Research Program in Software Performance

- **Aim to estimate (and optimise) performance of a design, running on a typical platform**
- **Methods that work with any development technique**
  - **automated model-building by angio traces**
- **Systematic characterization of software components**
- **Better abstract models**
  - **layered queueing for logical resources, tools**
  - **acceptance**
- **Exploit model results to optimise designs**
- **‘til now, most of my applications in distributed (network) systems**
  - **for instance, client-server systems**
  - **a little in embedded systems**

Workshop on Software and Performance (WOSP98)

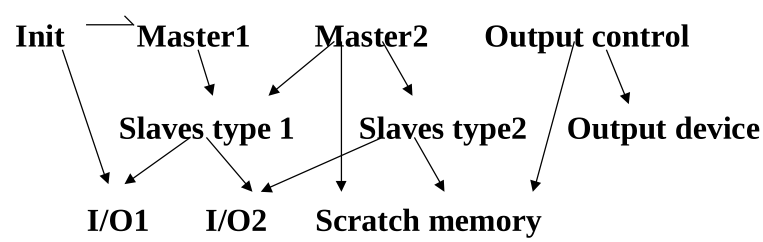
- **early October 98… proceedings available from ACM**
- **practices, case studies, methodology, tools**
- **WOSP is dedicated to introducing performance concerns into software engineering**
  - **"performance failures are the single most common cause of project failure"**
  - **methodology integration**
  - **professional practice**
  - **better tools and standards for thetechnical aspects of performance prediction**
- **…. watch for WOSP99 next Fall**

Resources and architecture

- ***Architecture* is structure**
- **Entities that make up a system, and their attributes and relationships**
- ***Resources* are machinery that makes the system go.**
  - **physical**
  - **logical**
- **They must be allocated**
- **They have limitations that may constrain performance.**

Example: heirarchical architectures

- **Heirarchy is a common architecture**
  - **levels; with responsibility types**
  - **in a level, roles with detailed responsibilities, object classes with properties, and concrete objects (different abstractions)**
- **architecture of roles, of object classes, of concrete objects**
  - **a complex master-slave architecture over classes:**



What is *software* architecture?

- **It is the top level of software design**
- **System organization**
  - **objects**
  - **interfaces**
  - **control and inheritance relationships**
- **Requires partitioning and allocation of functionality, identifying support functions**
- **Large scale architecture first, then recursively define subsystem architectures**

Software architecture classes

- **described by David Garlan and Mary Shaw at CMU**
  - **heirarchy**
  - **client-server (heirarchy with call-return)**
  - **master-slave (single boss)**
  - **pipeline**
  - **blackboard**

Resources

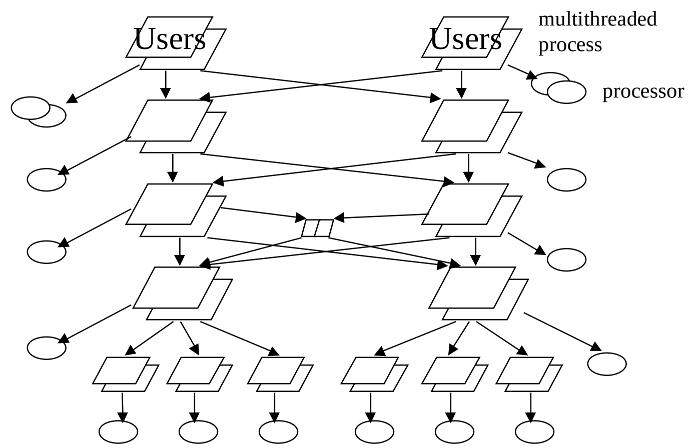
- **Physical devices:**
  - **CPUs, disks, printers, buses, memory**
  - **Interface devices (A/D, D/A, motor controllers, network interface devices, camera interfaces**
  - **Network connections**
- **Logical resources:**
  - **Critical sections, locks, buffers, copies of processes, threads**
  - **window flow control tokens**
- **a resource has a discipline that controls granting, and a holding time before release**
- **we conceptually model a resource as a token, signifying the *capability* to use the resource**

Logical resources

- How can I detect the use of a logical resource?
- *Criterion:* any entity that has a request procedure that can make processes wait, and a limited supply that controls granting the request, is a logical resource. Examples:
  – critical section entry request controlled by a semaphore
  – lock request controlled by a lock manager
  – buffer request controlled by a buffer manager
- synchronization is not in this category, though it can make you wait
- there are no infinite resources
  – e.g., if the number of buffers is not controlled, memory may be exhausted; similar for threads.
  – however, infinite resources may be used in models to show a constraint removed

Example with complex process *thread* resources

- resource architecture is embedded in the software design
- resources are held simultaneously by a job to do work



Processes and *"threads"*

- in general a thread is just a lightweight process
- here, the name thread represents a "clone" of a process, such that the clones share the requests. They can be implemented in various ways:
  - (1) forked UNIX processes
  - (2) "lightweight" kernel or user threads such as p-threads, Mach threads, Solaris threads, many others
    - – threads share data and context, and interact via mutex, fork/join, signalling,
    - – lower (but not zero) cost of context switch
  - (3) user-managed data on the current set of operations, sometimes called "asynchronous servers" (recent paper by Niklaus Wirth) ... here, called *"data threads"*
    - – lowest context switch cost (change pointer to data)

Factors governing the use of threads

- Lightweight threads and data threads are more dangerous than operating system processes, since you give up the data protection erected between processes
- Less overhead
- Cheaper communications between threads
- More complicated programming
  - – for data threads, the programmer has to manage all the context maintenance in tables of data for different operations begun but not yet completed, and waiting for some further input
  - – lightweight threads are meant as a compromise
- May involve user-level thread scheduling
  - – so-called "kernel threads", eg in Solaris, OS2 and NT, are scheduled by the kernel

“Asynchronous” or “non-blocking” servers

- **widely used for extra high performance**
- **built with *data threads***
    - **one single-threaded task executing in a loop**
    - **task is always ready to receive any message, when it reaches the head of its loop**
    - **if the message is a reply to a previous request it made, it pulls out the context of the request from data tables, and executes for it**
- **if the context table has a finite limit in size, this is the number of threads**
    - **if no finite limit, the server is infinite-threaded**
    - **this gives linear software and avoids task resource limits and blocking.**
- **however notice that individual data threads do block...**

Performance interpretation of threads

- **time to complete one request is a *thread service time***
    - **if unbounded number of threads, the task has no resource limit**
    - **if bounded, the threads are a resource limit (if large enough number, the limit can be ignored)**
    - **time may depend on which service was requested**
- **threads compete for other resources (just like tasks)**
    - **processor, I/O**
    - **a mutex between threads**

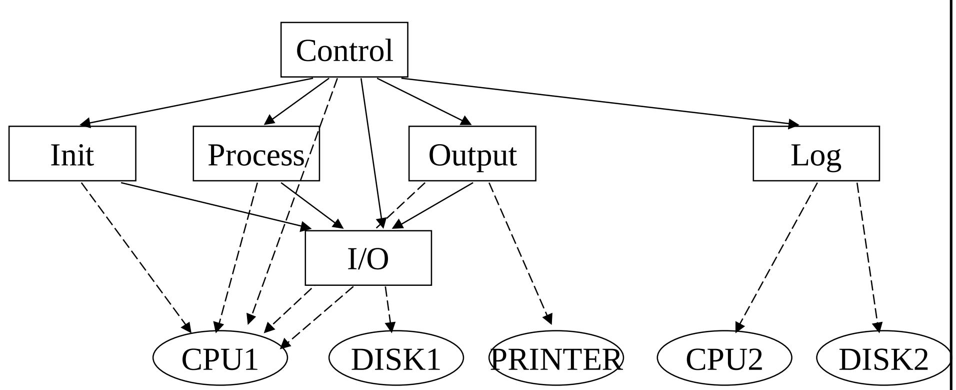
Consequences of resource architecture

- **the relationships and holding times of resources control behaviour and performance:**
  - **deadlock due to mutual waiting**
  - **performance limitation**
- **deadlock is well studied for a single category of resources…**
  - **for locks,**
  - **for semaphores,**
  - **for single threaded processes…**

  but not globally for all resources
- **performance is well understood for *flat* resource architectures, but not in general**
  - **flat == queueing networks, classical scheduling**
  - **general case involves <u>*simultaneous resource possession*</u>.**

How resource architecture emerges from a software design

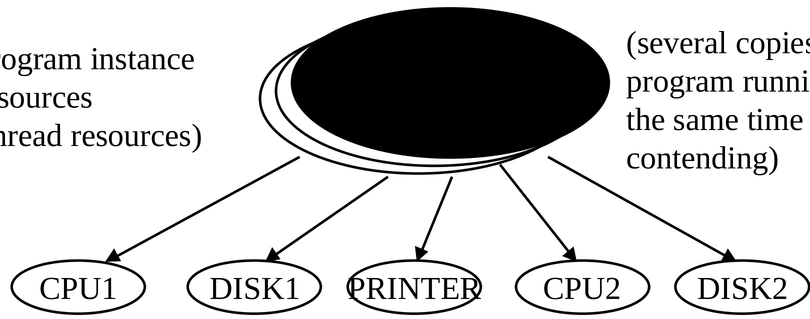
- **here is a software system with one-at-a time resources…**
  - **boxes are large subsystems**



One-at-a time resources give simple architecture...

Program instance resources (thread resources)

(several copies of the program running at the same time and contending)



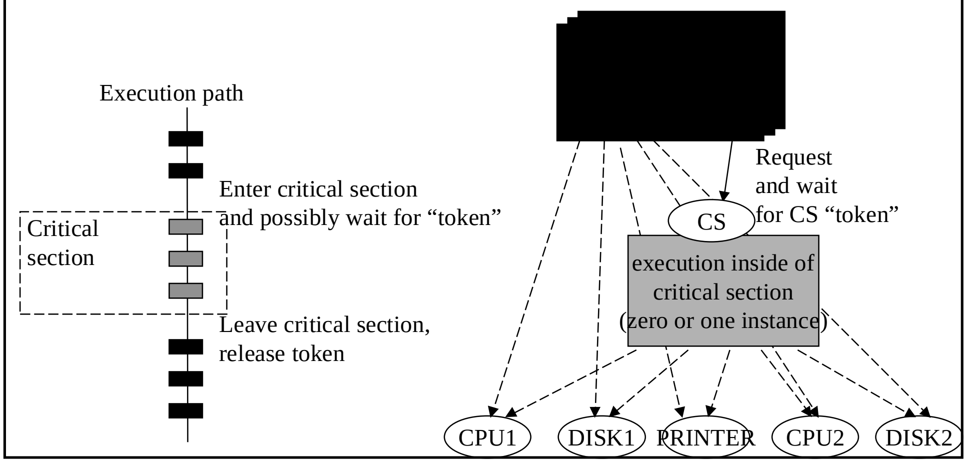
- users, plus flat set of other resources
- for understanding queue-based contention resolution, the software gives the parameters of a queueing model with just the devices in the model (classical performance model)
- for deterministic deadline systems, contention is resolved in the scheduling of activities at each resource

Now suppose the software has a critical section

- **this “resource” is created in the software.**
  - **think of the critical section as controlled by a token.**
  - **the process that holds the token can execute in the critical section**
- **only one user of the resource (token) at a time**
- **others must wait**
  - **queue**
  - **timeout and retry**
- **while holding the resource, an execution uses other resources and does other things**

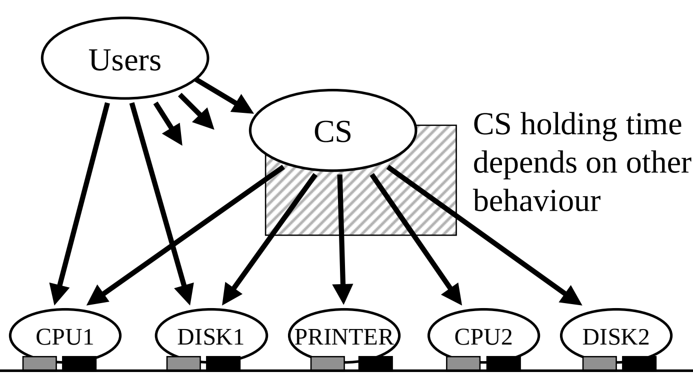
How software design introduces a logical resource (a critical section)

- **A critical section example in the same program….**



Resource architecture results

- **Dependency between CS resource and devices, induced by the software behaviour…..**
- **CS can limit performance**



Serious consequences: logical bottlenecks

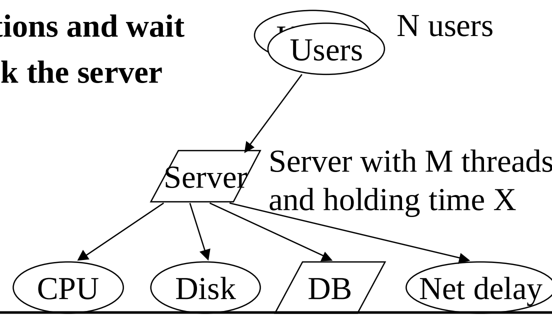
- **Example of an internet based server**
- **server threads may be a *logical bottleneck***
- **thread holding time**

    X = cpu demand + cpu wait

    + disk operations and wait

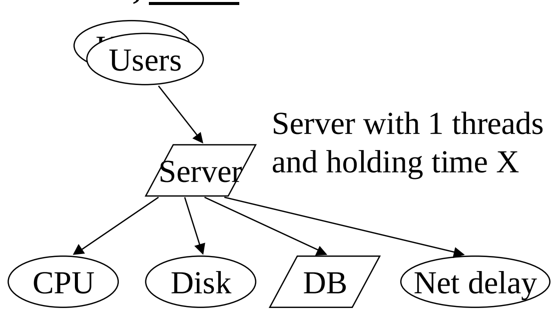
    + lower server operations and wait

    + net delays that block the server
- **throughput limit M/X for server thread saturation**
- **what is X? it may be dominated by *Net delay***

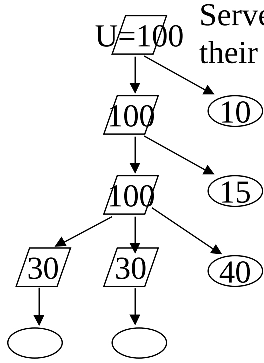


Simplify: single threaded server

- **X is now just the sum of the demands, since the server is a critical section**
    - **net delay is part of the service time!!**
    - **so much for fast execution!!**
- **so as load increases, <u>Server</u> saturates first**



Multiple levels: saturation spreads upwards

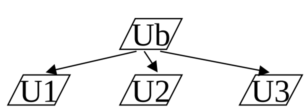


Servers labelled with their utilizations

- **Server utilization =**
  - **the fraction of time it is busy and blocked (one thread)**
  - **thruput x (thread service time) if many threads**
- **The third level server is saturated because it blocks and waits for other servers**
- **it saturates the layers above it**

Software bottleneck: general properties

- **Typically the bottom resource is a processor... a bottleneck there is of the familiar hardware type**
  - **a saturated processor can "push up" on higher resources and make them saturated too, by blocking them**
  - **this is still a hardware bottleneck**
- **A pure Software Bottleneck appears only at a higher resource in the layering; lower resources are not responsible**
  - **typically associated with a "fan-out"... the higher resource delays are contributed by two or more lower resources,**
  - **BS >1 and U_b ~ 1 (Neilson..., IEEE Trans on Soft Eng'g 1995)**



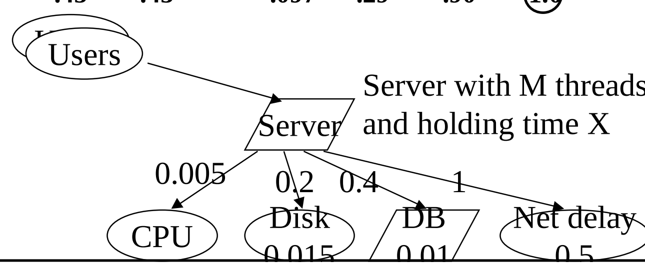
U_b = utilization of task or resource b

- **Bottleneck strength measure at task b: $BS_b = U_b / \max_i U_i$**

Results for the web server with net delay

N users	500	500	500	500	2000	2000	2000	2000
M threads	10	30	100	inf	10	30	100	inf
X server	.512	.52	.52	.52	.512	.515	.55	4.99
f thruput	19.5	58.2	90.6	90.6	19.5	56.7	180	200
W user wait	20.6	3.6	0.51	0.5	97.6	29.4	6.1	5
U server	10	30	47	47	10	30	100	1000
U net	9.7	29.1	45.3	45.3	9.7	29.1	90.2	100
U CPU	.097	.29	.45	.45	.097	.29	.90	1.0

N Users with
a thinking time
of 5 sec.



Thoughts on logical resources

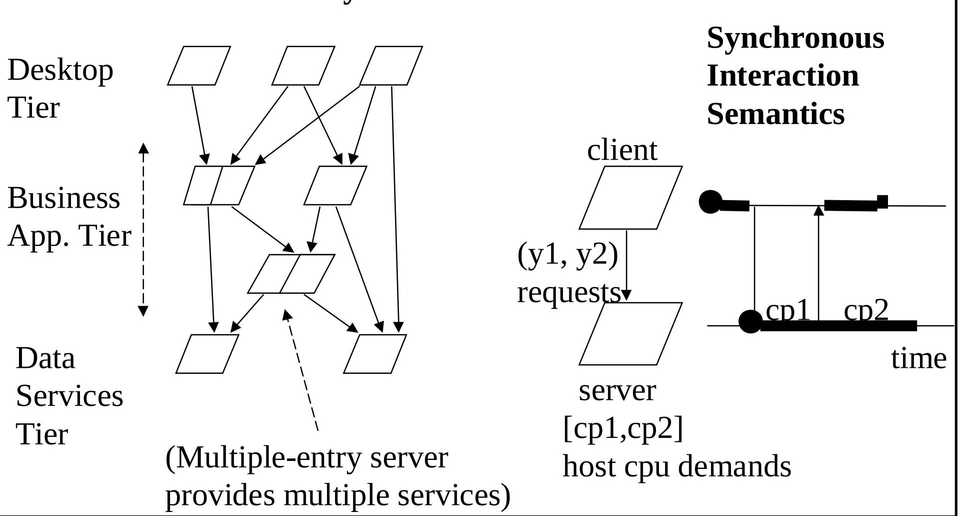
- **Reduced holding time improves capacity**
  - **early release of a client, so the server does delayed work after, often pays off!**
- **Multiplicity relieves waiting**
  - **more buffers, threads**
  - **memory vs performance tradeoff**
  - **balance… provide themwhere they will do good**
  - **evaluate! Layered modeling is one way.**
- **Multi-threading a process has a programming cost also. *Threads* take many forms in practice**

Formal performance models

- **Layered queueing network**
  - **top level entities are drivers, such as user processes or timers that trigger execution**
  - **bottom level entities are pure servers that serve requests from users, such as CPUs**
  - **mid-level entities accept service requests, and while serving them, issue requests to lower servers (and wait for completion)**
- **each server has a request queue and a service discipline**
- **Other models, e.g. simulation of detailed resource interactions, also work...**

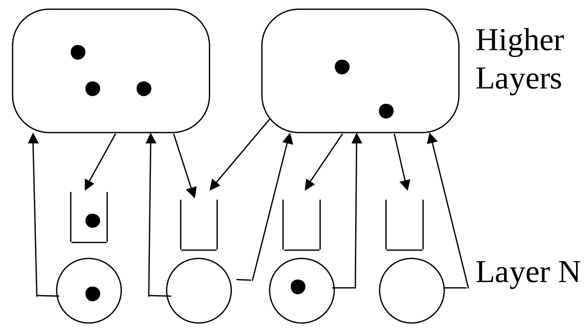
Layered Architecture Notation for LQNs

Generic Three-Tier
Client Server System



Tool Support: LQNS Solver Structure
"Active Server" Queueing Sub-Model for each Layer

"Delay Server" with tokens that represent clients or active servers from layers above



Queue and server tokens represent requests from upper layer servers

Servers represent server tasks at layer N:
- Service times of the servers include delays at lower layers, including processors
- Servers may be non-standard (two phase!)

Characterizing Architectures: Patterns and Properties

- **to understand which architectures are good for which situation, we would like to classify them into families, and explore their performance as parameters change**
- **"patterns" show certain important properties or represent familes of applications**
  - **e.g. "Tower" pattern shows depth of dependency only**
    - **"Funnel" pattern shows many high-level tasks funneling into fewer at lower levels**
    - **"Pipeline" is common too.**
- **consider concrete architectures first**
  - **properties of architectures of classes and roles are more abstract**
- **evaluation: considering performance here.**

Pipelines

- **commonest form of effective concurrency, one of the basic software architectures identified by Garlan and Shaw**
- **divide the work by function and pass it on through a series of tasks**
  - **UNIX pipe: re-use basic operations implemented by tasks**
  - **business report generation**
  - **specialized attached processors**
  - **attached comunications processors *may* be pipelined**
    - **one for compression**
    - **one for network protocol, typically a board (NAB)**
    - **one for link or media access**

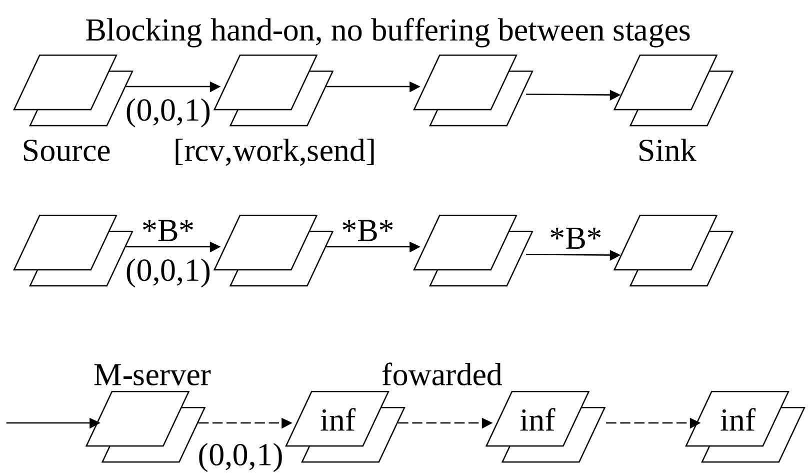
Pipeline

- **Three phases (input, execute, hand-on).**
- **Phase 3 deterministic (one message)**
- **RV Pipe, BB Pipe, FC Pipe**

RV Pipe
Pure RV hand-on

BB Pipe:
Bounded buffer hand-on

FC Pipe:
Flow-controlled



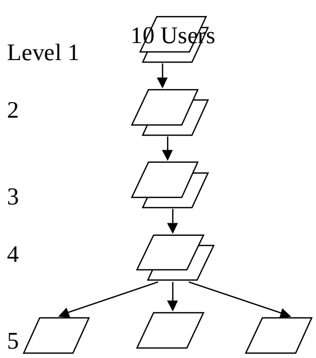
(max M tokens in the pipe at once) All deterministic phase-3 for hand-on

Pipelines and bounded buffers: Results

- **2, 4, and 10-stage pipeline**
- **execution demand 1 unit for work, 0.05 units to receive input and the same to send on output (thus, [0.05, 1.0, 0.05])**
- **so the ideal throughput f = 1/1.10 = 0.91/sec for 100% utilization**
- **B = 0 to 18 buffer places between stages**

	Ten-stage Pipeline…………		Four-stage	Two-stage
B	**Thruput f/sec**	**Proc'r Ut'n Range**	**Thru f/sec**	**Thru f/sec**
0	**0.40**	**0.42 - 0.45**	**0.49**	**0.63**
1	**0.50**	**0.52 - 0.56**	**0.60**	**0.72**
2	**0.5**	**0.63 - 0.66**	**0.66**	**0.78**
4	**0.8**	**0.73 - 0.75**	**0.75**	**0.83**
6	**074**	**0.76 - 0.81**	**0.78**	**0.85**
8	**0.76**	**0.80 - 0.85**	**0.80**	**0.88**
10	**0.79**	**0.83 - 0.88**	**0.83**	**0.88**
18	**0.83**	**0.87 - 0.93**	**n/a**	**n/a**

Tower and threading levels



(single servers
at the bottom)
(1 sec demand at each server, one
request to each lower task)

Throughput *f resp/sec* depends on thread levels

.........mi threads at level i, for i = 2, 3, 4

Threads	Thru	Utilizations
(m2, m3, m4)...	**f**	**(u2,u3,u4)**
(1, 1, 1)..........	**0.166,**	**(1., .83, .67)**
(2, 1, 1).........	**0.200,**	**(0.96, 1, 0.8)**
(3, 2, 1).........	**0.223,**	**(2.9, 1.64, 0.89)**
(6, 5, 4).........	**0.475,**	**(5.5, 3.9, 2.75)**
(10,10,10).......	**0.65,**	**(9.3, 7.8, 6.2)**

(utilization=mean busy threads)

Conclusions about the Tower cases...

- **a "Rule-of-Thumb" may be defined for threads:**
  - **If mi = threads of object i**
  - **Then set mi = mi* = sum of threads of *servers* of object i**
- **this works bottom up. For the Tower we have, this gives:**
  - **m4 = 4 (processor 4 plus three bottom servers)**
  - **m3 = 5 (processor 3 + m4)**
  - **m2 = 6 (processor 2 + m5)**
  - ***notice that additional threads beyond this do pay off! This is because there are 10 Users...***
- **on the other hand if we had only 5 Users, then m2 = 5 is all the threads that can be kept busy. This extends to:**
  - **If threads of clients of object i are limited to a total of T, and object i has only one phase, then mi =min(T, mi*)**

Tower with Second Phases

- **each service at layer 2, 3, 4, 5 has a first phase 1-α sec, and a second phase of α sec. This greater concurrency and faster release of the client in all interactions should increase capacity…**

[Ph1, Ph2]	Throughputs f in responses/sec (accurate to ~ +/- 0.02)		
host demand	m = (10,10,10)	m = (6,5,4)	m = (1,1,1)
[1, 0]	0.65	0.475	0.166
[0.83, 0.17]	0.67	0.53	0.19
[0.67, 0.33]	0.68	0.57	0.21
[0.50, 0.50]	0.68	0.55	0.24
[0.33, 0.67]	0.66	0.55	0.25
[0.17, 0.83]	0.66	0.56	0.26
[0,1]	0.64	0.53	0.27
ref X. Lu patt96 p 40	least impact	medium impact	most impact

- **more impact when threads are a little scarce, and over the first 30% or so**

Tower with competing load at one Level

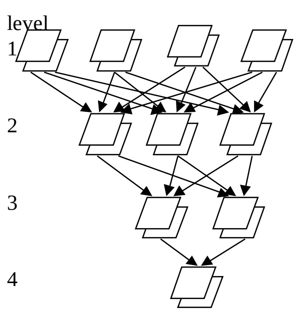
- Suppose the processor of T2 (task at level 2) also carries an externally imposed load (arrival rate f_{ext}, service time 1.0 sec),
  - this degrades the responses of the Tower.
- If the external load is at one of the three level 5 servers, the effect is much worse

Ext rate	Processor	Tower throughput f/sec	
f_{ext}/sec	utiliz’n by ext.	(Load at T2)	(Load at level 5)
0	0	0.475	0.475
0.2	20%	0.445	0.437
0.4	40%	0.435	0.415
0.6	60%	0.427	0.383

ref X.Lu patt96 p 30

Copyright C. M. Woodside 1998

Funnel



- with 100 users in total across the four groups, and user time (think time) = 10 sec

Threads	Throughput	(Ref Brunet
(m2, m3, m4)	f/sec	patt96 p 54-5)
(1, 1, 1)	6.0	
(2, 2, 1)	8.6	
(3, 2, 1)	8.9	
(4, 5, 1)	9.4	

- the example is starved by long user think time, for f=<N/(Z+D) gives f =< 10/sec
- horizontal imbalance.. little effect
- vertical balance: very bad to concentrate load at level 4 (single bottleneck)

Conclusions

- Resource architectures are a by-product of software architecture and design
- though little studied, they deeply affect performance
- there are some subtle aspects
  - identifying resources
  - identifying the workload done within the context of a resource
- simple back-of-envelope resource capacity calculations don't work
  - the holding time of higher-level resources is not easy to know
  - full modelling is needed
- implications for hard real-time systems are still to be understood

Additional Reading (1)

- Franks 95: G. Franks, "Traffic Dependencies in Client-Server Systems and Their Effect on Performance Prediction", pp 24--33, Proc. IEEE International Computer Performance and Dependability Symposium, Apr 95
- Hrischuk et al 95: C. Hrischuk, J.A. Rolia, C.M. Woodside, "Automatic Generation of a Software Performance Model Using an Object-Oriented Prototype, Proc. of Third Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '95), Durham, NC, pp399--409, Jan 1995.
- Majumdar et al 91: S. Majumdar C.M. Woodside J.E. Neilson, D.C. Petriu, "Performance Bounds for Concurrent Software with Rendezvous", Performance Evaluation, v 13, pp 207--236, 1991
- Miernik et al, 88: J.W. Miernik, C.M. Woodside, J.E. Neilson, D.C. Petriu, "Performance of Stochastic Rendezvous Networks with Priority Tasks", Proc. of Int. Seminar on Performance of Distributed and Parallel Systems, pp 501--515, December 1988
- Petriu 91: D.C. Petriu, C.M. Woodside, "Approximate Mean Value Analysis for Markov Model of Software Client/Server Systems", Proc. Third IEEE Symposium on Parallel and Distributed Processing, Dallas, pp 322-329, December 1991.
- Neilson et al 95: J.E. Neilson, C.M. Woodside, D.C. Petriu S. Majumdar, "Software Bottlenecking in Client-Server Systems and Rendezvous Networks", IEEE Trans on Software Engineering, v 21, n 9, pp 776--782, Sep 95
- Woodside 89: C.M. Woodside, "Throughput Calculations for Basic Stochastic Rendezvous Networks", Performance Evaluation, vol 9, no 2, April 1989
- Woodside et al 95: C.M. Woodside J.E. Neilson, D.C. Petriu, S. Majumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-like Distributed Software", IEEE Transactions on Computer, v 44, n 1,pp 20--34, Jan 1995.

Additional Reading (2)

- **Woodside 86: C.M. Woodside, E. Neron, E.D.S. Ho, B. Mondoux, "An ‘‘Active-Server’’ Model for the Performance of Parallel Programs Written Using Rendezvous", J. Systems and Software, pp 125 - 131, 1986**
- **Rolia 88: J.A. Rolia, "Performance Estimates For Systems With Software Servers" Proceedings VIII SCCC International Conference on Computer Science, pp 25--43, July 1988**
- **Rolia and Sevcik 95: J.A. Rolia K.C. Sevcik, "The Method of Layers", IEEE Trans. on Software Engineering, v 21, no 8, pp 689--700, August 1995**

PLUS...

- **Proceedings of the first Int. Workshop on Software and Performance, Santa Fe, October 1998 (order from ACM in New York)**
  - **Selected papers to appear in the IEEE Transactions on Software Engineering**
- **Web site: www.sce.carleton.ca/rads is a site for the entire group, with an introduction to layered modeling and even a web interface to the modelling tool.**
  - **www.sce.carleton.ca/faculty/woodside**