Adaptive Overload Control for Busy Internet Servers

Matt Welsh and David Culler
Intel Research, Berkeley
And University of California Berkeley

Presented by: Lev Kaufman

Matt Welsh

- He is currently a visiting researcher at Intel Research, Berkeley and will be joining the faculty at Harvard University in July, 2003. He completed his Ph.D. in Computer Science at U.C. Berkeley in August 2002.
- research interests: operating systems design, distributed systems, networking, parallel computing, language and compiler support, and Internet service platforms

David Culler

- He is currently a visiting researcher at Intel Research, Berkeley and is a Professor in Computer Science at UC Berkeley.
- research interests: wireless networking, Internet services and parallel computing

Motivation for new overload control techniques

- Internet services are vital resource
- Internet services increasing in scale and complexity
- Much prior research on static web serving
- Dynamic servers different then static servers
  - Cannot cache and replicate dynamic content
  - Resource requirements for dynamic load difficult to estimate
- Huge variation in demand
  - CNN during 9/11, Peak load was 20 times expected peak

Overprovisioning not sufficient

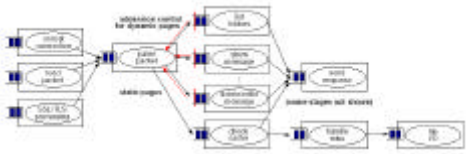
- Cannot predict maximum catastrophic load
- Cannot afford to have 100 times or greater extra equipment unused

Design Theory

- Services should be designed from ground up to detect and respond intelligently to overload conditions
- Overload management should be explicit in the programming model
- Services should have fine grained control of resources to respond to heavy load intelligently

Staged Event-Driven Architecture (SEDA)

- Network of event-driven stages
- Stages connected with explicit request queues
- Admission control to each queue
- All of the above enabling focused control of request flow



Structure of Arashi SEDA-based email service

SEDA benefits

- Enables overload control on a per node basis
- Supports large scale concurrency demands of modern server environment
  - Paper states that “general purpose threads are unable to scale to the large numbers required by busy Internet services”
  - This may not be true soon, as the Native POSIX Thread Library for Linux (new in the 2.5 Linux kernel) may be much more capable as suggested by this quote “In one instance starting and stopping 100,000 threads formerly took 15 minutes; this is now takes 2 seconds.”
    http://people.redhat.com/drepper/nptl-design.pdf.

SEDA benefits (cont.)

- Exposure of request stream
  - Observe and control request stream
  - Allows explicit rejection of requests
- Focused, application specific admission control
  - Can control certain types of request in application specific manner
- Modularity and performance isolation
  - Allows stages to be insulated from each other

User rejection versus unacceptable response times

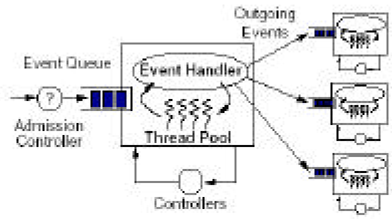
- SEDA can send explicit rejection messages rather then just have TCP connections retry (as in Apache)
  - Explicit indication of overload is different from traditional model of design which treats overload as an exceptional case
- SEDA can reject a request at a certain stage without rejecting entire request
  - Finer grained control

Performance Metrics

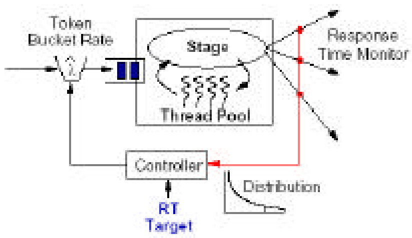
- 90th percentile response time
  - Realistic
  - Intuitive
- Average or Maximum response time
  - Fails to represent “shape” of curve
- Throughput
  - Depends on network
  - Little relation to user perception

SEDA Stage design

- Uses 90th percentile response time as metric
- Structure
  - Queue
  - Response time monitor
  - Admission Controller
    - Uses weighted moving average of response time
  - Token Bucket Rate
    - Controlled by controller
    - Increases additively, decreases multiplicatively
  - Thread pool and Event Handler
    - Handles each request taken off the queue



A SEDA Stage



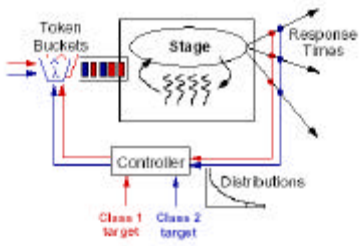
Response time controller design

Parameter	Description	Default value
$nreq$	# reqs before controller run	100
$timeout$	Time before controller run	1 sec
α	EWMA filter constant	0.7
err_i	% error to trigger increase	-0.5
err_d	% error to trigger decrease	0.0
adj_i	Additive rate increase	2.0
adj_d	Multiplicative rate decrease	1.2
c_i	Weight on additive increase	-0.1
$rate_{min}$	Minimum rate	0.05
$rate_{max}$	Maximum rate	5000.0

Parameters used in the response time controller

Additional granularity of control

- Class based differentiation
- Service degradation
  - Can be used in conjunction with Class based differentiation



Multiclass overload controller design

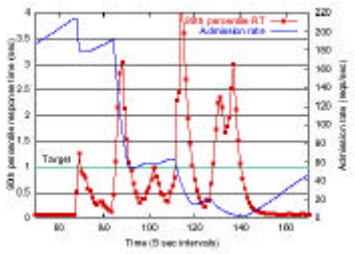
A SEDA application, Arashi

- Web based email service similar to Hotmail
- Coded in Java
  - Claims performance competitive with similar application coded in C

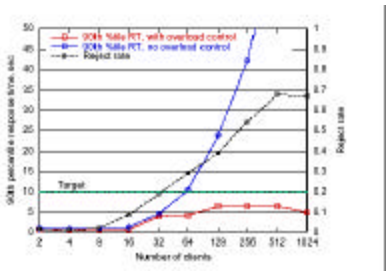
Screenshot of the Arashi email service

Performance evaluations

- Overload control much better then no overload performance in terms of response time.
- But leads to many rejected requests
- Authors state: “We claim that giving 20% of the users good service and 80% of the users some indication that the site is overloaded is better than giving all users unacceptable service.”



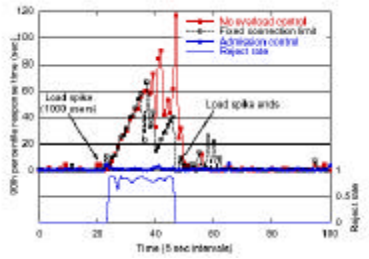
Overload controller operation



Overload control in Arashi

	16 users		*1024 users*	
Type	**No OLC**	**OLC**	**No OLC**	**OLC**
login	0.83 sec	0.59 sec	0.86 sec	3.84 sec
list folders	1.73 sec	0.57 sec	**365 sec**	5.75 sec
list msgs	2.37 sec	0.58 sec	**116 sec**	9.28 sec
show msg	0.70 sec	0.30 sec	**30.1 sec**	3.87 sec
delete	1.00 sec	0.28 sec	**11.3 sec**	6.85 sec
refile	1.00 sec	0.47 sec	**10.6 sec**	6.07 sec
search	8.17 sec	9.92 sec	**19.6 sec**	**18.1 sec**

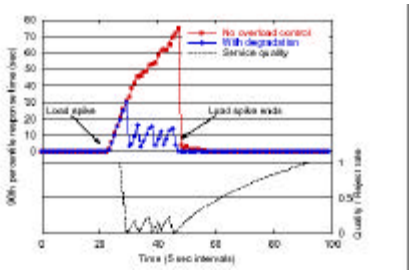
Breakdown of response times by request type for Arashi email service



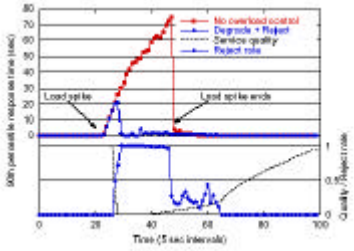
Overload control under a massive load spike

	Per-stage AC		*Single-stage AC*	
Type	**90th RT**	**Rejected**	**90th RT**	**Rejected**
login	2.07 sec	44.3%	1.00 sec	18.8%
list folders	8.11 sec	59.6%	3.97 sec	59.6%
list msgs	8.04 sec	47.1%	6.20 sec	53.7%
show msg	3.90 sec	23.1%	2.04 sec	49.1%
delete	4.86 sec	11.3%	3.26 sec	51.4%
refile	4.60 sec	10.4%	2.12 sec	54.7%
search	**22.2 sec**	0%	**18.9 sec**	53.3%

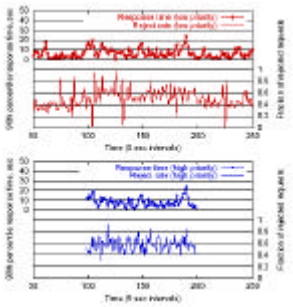
Comparison of per-stage versus single-stage admission control



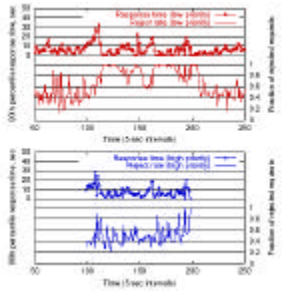
Effect of service degradation



Service degradation combined with admission control



Multiclass experiment without service differentiation



Multiclass service differentiation

Related work

- Resource containment
  - Static, with limits arbitrary and rigid
  - Not driven by monitoring and feedback of system
- Admission control
  - Many techniques based on fixed policies
  - Typically refuse connections
- Control theoretic approaches
  - Many have overly simplistic models
  - Hard to derive good models of system behavior
- Service degradation
  - Limited because many services are not degradable

Conclusions

- "Measurement and control are the keys to resource management and overload protection in busy Internet services."
- Arashi demonstrates how their system (SEDA) can achieve high quality 90th percentile response rates in overloaded conditions.

Future work

- Move past heuristic approach to controller design
- Work on global system reactions to overload not just reactions local to a given stage
- More direct measurement of per stage resource consumption