Scalable Internet Architectures

OmniTI / Operating at Scale

Apachecon US 2009

Who am I? @postwait on twitter

- Author of “Scalable Internet Architectures”
  Pearson, ISBN: 067232699X
- CEO of OmniTI
  We build scalable and secure web applications
- I am an Engineer
  A practitioner of academic computing.
  IEEE member and Senior ACM member.
  On the Editorial Board of ACM’s Queue magazine.
- I work on/with a lot of Open Source software:
  Apache, perl, Linux, Solaris, PostgreSQL,
  Varnish, Spread, Reconnoiter, etc.
- I have experience.
  I’ve had the unique opportunity to watch a great many catastrophes.
  I enjoy immersing myself in the pathology of architecture failures.

Topic Progression

- What is an architecture?
- What does it mean to run a (scalable) architecture?
- Scaling Techniques for
  - Static Content
  - Dynamic Content
  - Databases
  - Networks
- Techniques for decoupling services
- Bad Ideas

Architecture

OmniTI / the whole enchilada

Architecture / *what it is*

- architecture (n.):
  the complex or carefully designed structure of something.

  specifically in computing:
  the conceptual structure and logical organization of a computer or a computer-based system.

- Oxford American Dictionary

OmniTI

Architecture / *more than meets the eye*

- An architecture is all encompassing.
  - space, power, cooling
  - servers, switches, routers
  - load balancers, firewalls
  - databases, non-database storage
  - dynamic applications
  - the architecture you export to the user (javascript, etc.)

Architecture / *awareness is key*

- Not all people do all things.
- However...
    - lack of awareness of the other disciplines is bad
    - leads to isolated decisions
    - which leads to unreasonable requirements elsewhere
    - which lead to over engineered products
    - stupid decisions
    - catastrophic failures

OmniTI

Architecture / *running it all*

- Running Operations is serious stuff
- It takes ***knowledge***, ***tools***...
- but that is not enough.
- It takes ***experience***.
- And perhaps even more importantly...
- It takes ***discipline***.

OmniTI

Architecture / *knowledge*

- Read.
- Study.
- Leverage User Groups (SAGE,LUGs,OSUGs,PUGs,etc.)
- Participate in the community.

Architecture / *tools*

- Collaborate with colleagues.
- Try new tools.
- Write new tools.
- Know and practice your tools during the “good times”
  in order to make their use effortless during the “bad times”

Architecture / *tool theories*

"One only needs two tools in life:
WD-40 to make things go,
and duct tape to make them stop."

- George Weilacher

"Man is a tool-making animal."

- Benjamin Franklin

"Man is a tool-using animal."

- Thomas Carlyle

"Men have become the tools of their tools."

- Henry David Thoreau

"All the tools and engines on earth
are only extensions of man's limbs and senses."

- Ralph Waldo Emerson

Architecture / *my take on tools*

- Tools are just tools.
- They are absolutely essential to doing your job.
- They will never do your job for you.
- Tools will never replace experience and discipline.
- But tools can help you maintain discipline.

Architecture / *experience*

“Experience is what enables you to recognize a mistake when you make it again.”

- Earl Wilson

“Is there anyone so wise as to learn by the experience of others?”

- Francois Voltaire

“Good judgment comes from experience. Experience comes from bad judgment.”

- Proverb

“Judge people on the poise and integrity with which they remediate their failures.”

- me

Architecture / *discipline*

- Discipline is important in any job.
- Discipline is

  "controlled behavior resulting from training, study and practice."
- In my experience discipline is the most frequently missing ingredient in the field of web operations.
- I believe this to be caused by a lack of focus, laziness, and the view that it is a job instead of an art.
- As in any trade
  - To be truly excellent one must treat it as a craft.
  - One must become a craftsman.
  - Through experience learn discipline.
  - And through practice achieve excellence.

OmniTI

Architecture / *actually running it all*

- Okay, I get it.
- From day to day, what do I need to know?

Architecture / *version control*

- Switch configurations should be in version control.
- Router configurations should be in version control.
- Firewall configurations should be in version control.
- System configurations should be in version control.
- Application configurations should be in version control.
- Monitoring configurations should be in version control.
- Documentation should be in version control.
- Application code should be in version control.
- Database schema should be in version control.
- Everything you do should be in version control.

OmniTI

Architecture / *version control*

- And no... it doesn’t matter which tool.
- It’s not about the tool, it’s about the discipline to always use it.

(today, we use subversion)

Architecture / *know your systems*

- To know when something looks unhealthy,
  one must know what healthy looks like.
- Monitor everything.
- Collect as much system and process information as possible.
- Look at your systems and use your diagnostic tools
  when things are healthy.

Architecture / *management*

- Package roll out?
- Machine management?
- Provisioning?

- They tell me I should use Puppet.
- They tell me I should use Chef.
- well... I stick to my theory on tools:
  - *A master craftsman chooses or builds the tools he likes.*
  - *A tool does not the master craftsman make.*

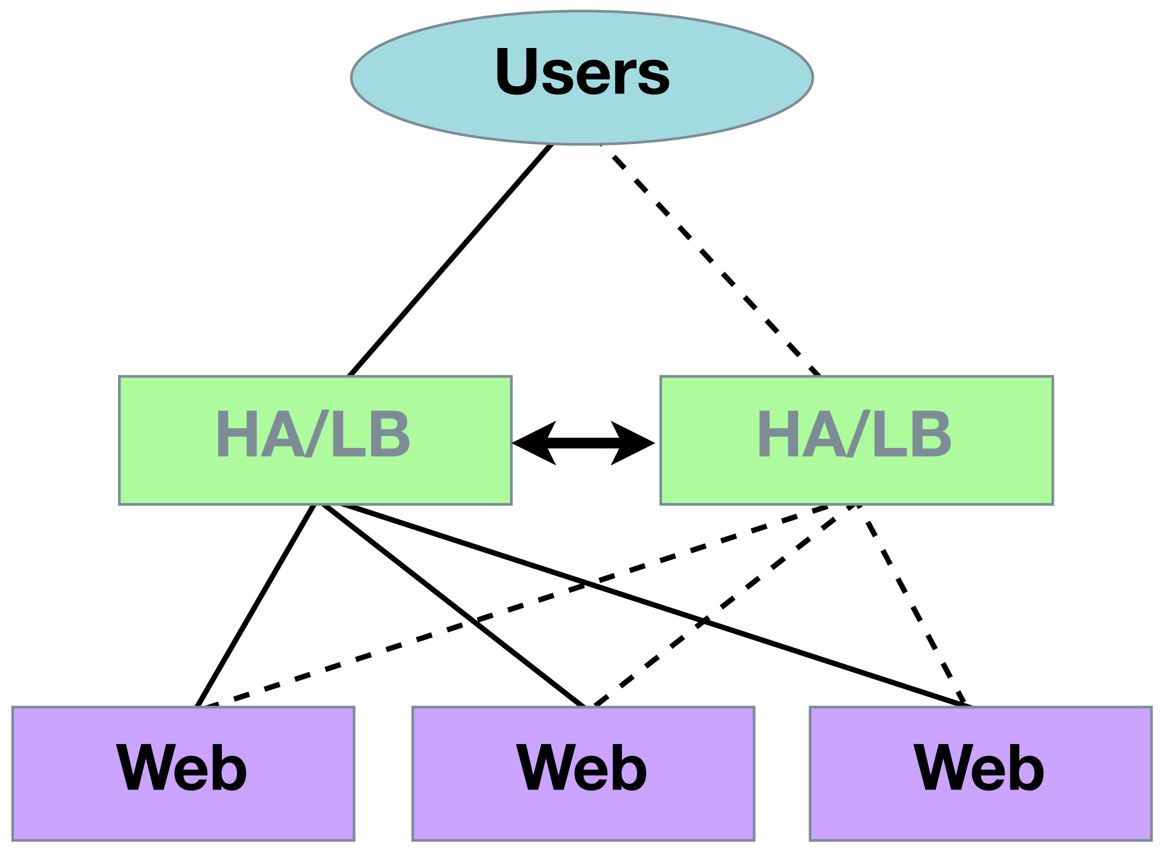
Static Content

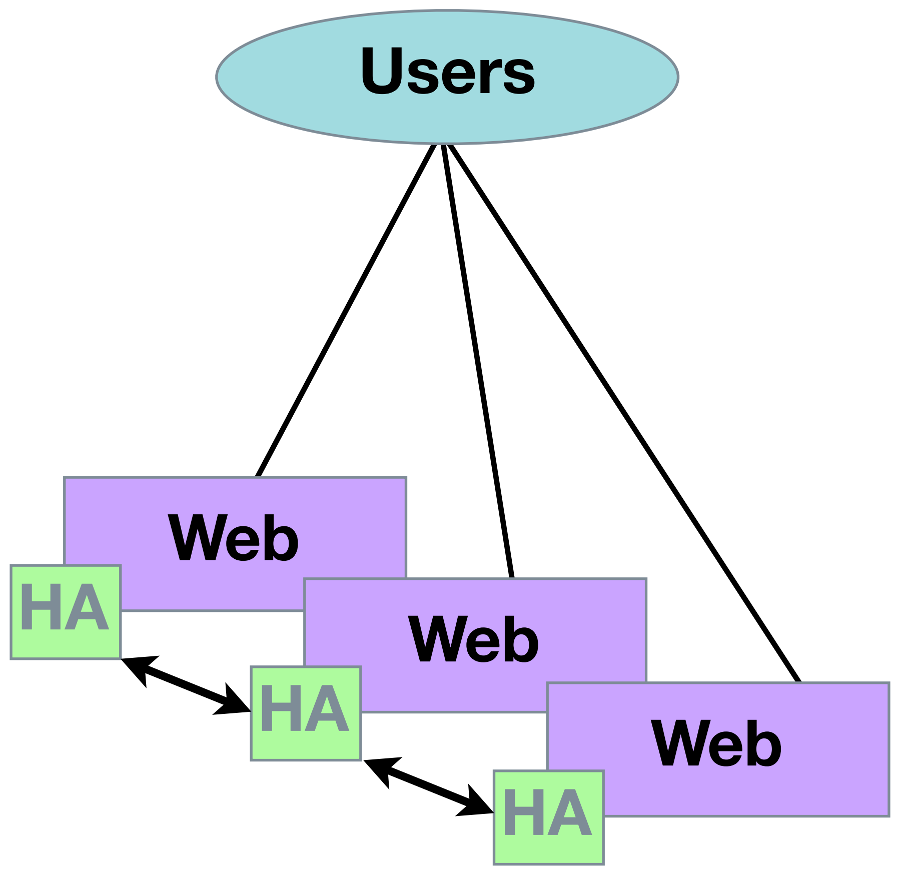
OmniTI / bits by the truckload

Techniques / *Static Content*

- Old tricks. Good games.
- Use Akamai… or a competitor… or build it yourself.

Content Distribution / *availability*





“White Paper” Approach

expensive, dedicated, single-purpose HA/LB devices

Peer-based HA

cheap and reusable commodity machines

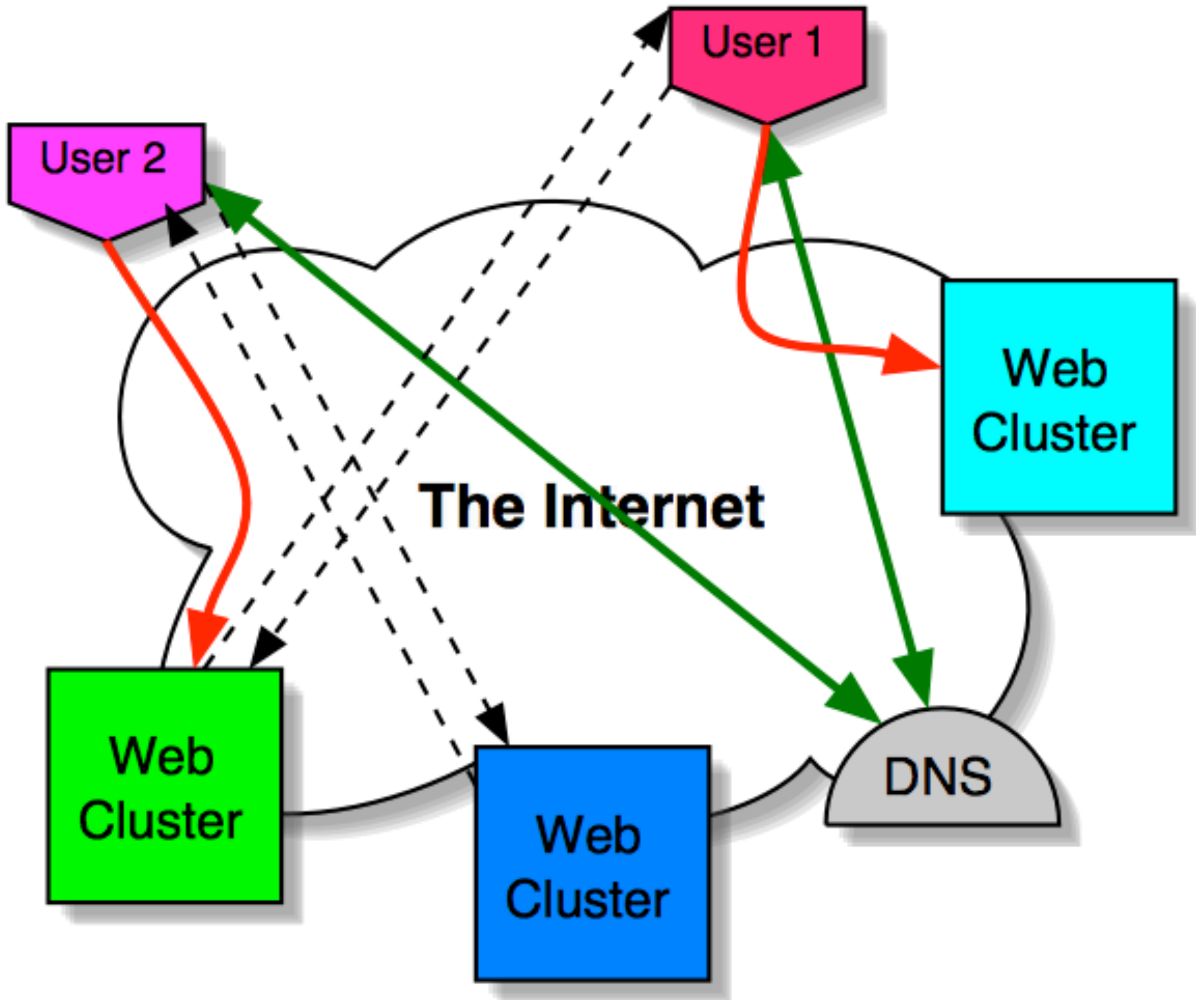
Content Distribution / *the stack*

- Setup a web server to host all your static content.
- Setup a handful of servers running a reverse proxy-cache:
Squid or ***Varnish*** or Apache/mod_proxy
- Make them redundant without a load balancer by using IP redundancy protocols:
VRRP, UCARP or ***Wackamole***
- simple, easy, scalable.

Content Distribution / *the network*

- Setup the same thing in multiple datacenters
- Each has its own set of IP address:
  - d.c.a.{11,12,13}
  - d.c.b.{11,12,13}
  - d.c.c.{11,12,13}
  - etc.

Content Distribution / *location*

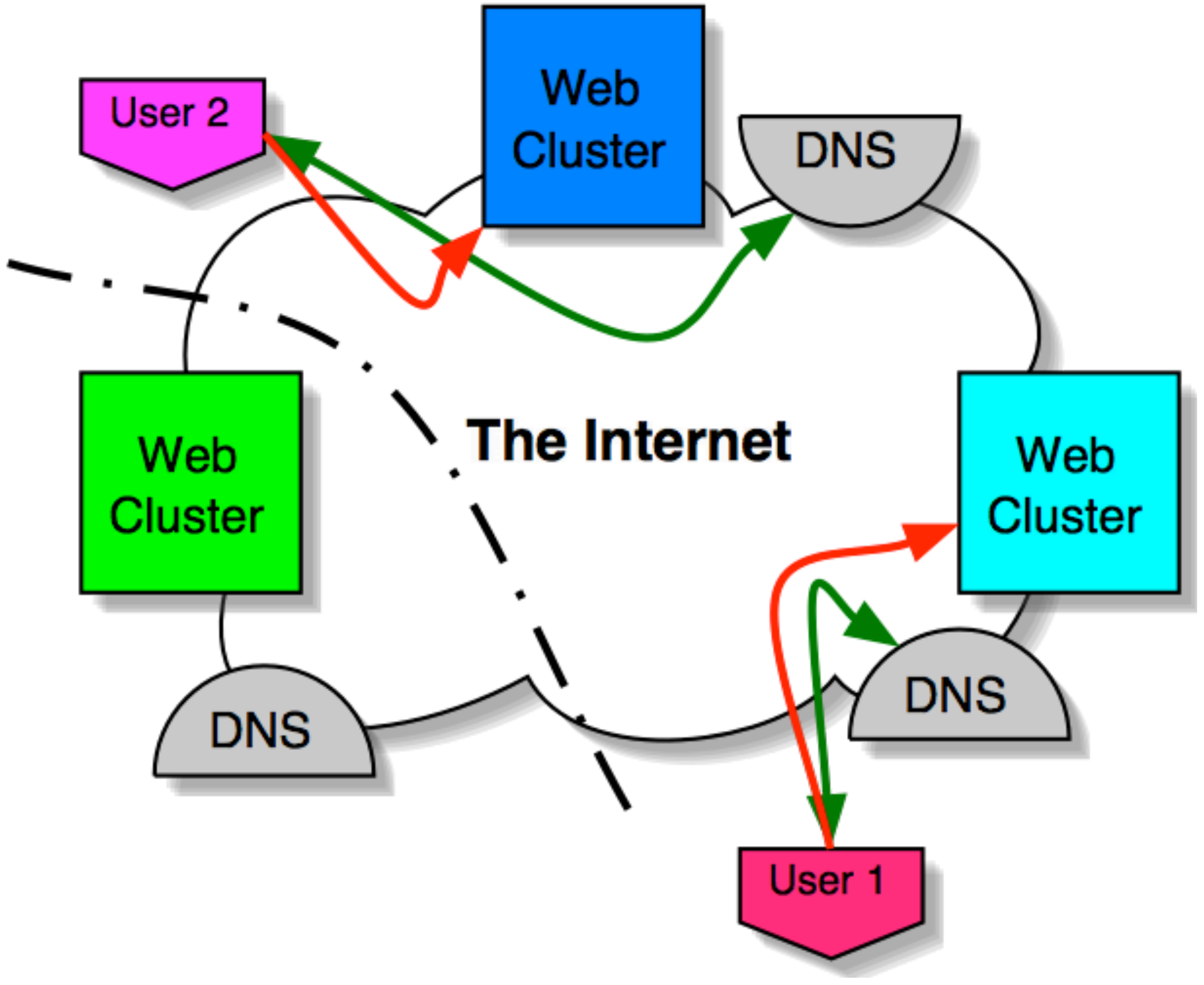


OmniTI

Content Distribution / *location seamlessly*

- Put a DNS server at each location behind the same uplink
  - each with the same IP address
  - announce that network from all data centers (using BGP)

Content Distribution / *global access*



Dynamic Content

OmniTI / keeping users interested

Techniques / *Dynamic Content*

“We should forget about small efficiencies,
say about 97% of the time:
premature optimization is the root of all evil.”

- Donald Knuth

“Knowing when optimization is premature defines the difference
between the master engineer and the apprentice.”

- me

Techniques / *optimization*

- Optimization comes down to a simple concept: “don’t do work you don’t have to.”
- It can take the form of:
  - computational reuse
  - caching in a more general sense
  - and my personal favorite:
    - ... avoid the problem, and do no work at all.

Techniques / *optimization applied*

- Optimization in dynamic content simply means:
    - Don’t pay to generate the same content twice
    - Only generate content when things change
    - Break the system into components so that you can isolate the costs of things that change rapidly from those that change infrequently.

Caching / *real world example*

- News site
  - News items are stored in Oracle
  - User Preferences are stored in Oracle
  - Hundreds of different sections
  - Each with thousands of different articles

- Pages:
  - 1000+ hits/second
  - shows personalized user info on EVERY page
  - front page shows top N_F articles for forum F (limit 10)

Caching / *the approach*

- Oracle is fast enough
  - why abuse Oracle for this purposes?
  - surely there are better things for Oracle to be doing
- Updates are controlled
  - updates to news items only happen from a publisher
  - news update:read ratio is miniscule
  - user preferences are only ever updated by the user

Caching / *articles*

- Article publishing
  - sticks news items in Oracle

- The straight forward way
  - http://news.example.com/news/article.php?id=12345
  - page pulls user prefs from cookie
  - (or bounces off a cookie populator)
  - page pulls news item from database

- I hate query strings
  - I like: http://news.example.com/news/items/12345.html

```
RewriteRule ^/news/items/([^/]*).html$ /www/docs/news/article.php?id=$1 [L]
```

Caching / *articles cached*

- We pull the item that is likely to never change
  - cheaper if the page just hard coded the news item
  - writing the news article out into a PHP page is a hassle
  - ... or is it?
- Have the straight forward page cache it
  - /news/article.php writes /news/items/12345.html
  - as a PHP page that still expands personal info from cookie, but has the news item content statically included as HTML.

```
RewriteCond %{REQUEST_FILENAME} ^/news/items/([^/]*).html
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^/news/items/([^/]*).html$ /www/docs/news/article.php?id=$1 [L]
```

OmniTI

Caching / *articles cached*

- Run a cache invalidator on each web server
  - connects to Spread as a subscriber
  - accepts /www/docs/news/items/####.html deletion requests
  - accepts full purge requests
- Article publishing
  - stash item #### in Oracle (insert or update)
  - publish through Spread an invalidation of ####
- Changing the look of the article pages
  - change article.php to have the desired effect
  - (and write the appropriate php cache pages)
  - publish through Spread a full purge
- If I had to do it again, I'd use a message queue instead of Spread.

Caching / *the result*

- All news item pages require zero DB requests
  - the business can now make your life difficult by requesting new crap on these pages that can’t be so easily cached
- Far fewer database connections required
  - all databases appreciate that (Oracle, MySQL, Postgres)
- Bottleneck is now Apache+mod_php
  - crazy fast with tools like APC
  - inherently scalable... just add more web servers
  - room for more application features

Data Management

OmniTI / remembering something useful

Techniques / *Databases*

- Rule 1: shard your database
- Rule 2: shoot yourself

Databases / *second try*

- Horizontally scaling your databases via sharding/federating requires that you make concessions that should make you cry.
- shard (n.)
  a piece of broken ceramic, metal, glass, or rock typically having sharp edges.
- sharding (v.)
  dunno... but you will likely wound yourself and you get to keep all the pieces.
- But seriously...
  - databases (other than MySQL) scale vertically to a greater degree than many people admit.
  - if you must fragment your data, you will throw away relational constraints. this should make you cry. cry. cry hard. cry some more. then move on and shard your database.

Databases / *vertical scaling*

- Many times relational constraints are not needed on data.
- If this is the case, a traditional ***relational*** database is unnecessary.
- There are cool technologies out there to do this:
  - “files”
  - CouchDB
  - cookies
- Non-ACID databases can be easier to scale
- Vertical scaling is achieved via two mechanisms:
  - doing only what is absolutely necessary in the database
  - running a good database that can scale well vertically

Databases / *horizontal scaling*

- Okay... so you really need to scale horizontally.
- understand the questions you intend to ask.
- make sure that you partition in a fashion that doesn’t require more than a single shard to answer OLTP-style questions.
- If that is not possible, consider data duplication.

Databases / *an example*

- private messages all stored on the server side
  - individuals sends messages to their friends
  - an individual should see all messages sent to them
- Easy! partition by recipient.
  - either by hash
  - range partitions
  - whatever

Databases / *an example complicated*

- now users must be able to review all sent messages.
- Crap!
    - our recipient-based partitioning causes us to map the request across all shards to answer messages by sender.
- In this case:
    - store messages twice... once by recipient and once by sender
    - twice the storage, but queries only hit a single node now

Databases / *an example unwound*

- Partitioning data allows one to reduce the dataset size on each node.
- You might just cause more problems than you've solved.
- Complicated (or even simple) queries become a pain
  if they don't align with your partitioning strategy.
- Partitioning like this is really a commitment. You lose much of the power of your relational database and complicate what were once easy problems.
- Sometimes you have to do what you have to do.
  Don't make the concession until you have to.

OmniTI

Databases / *take care*

- Multi-master replication is simply not ready these days.
  - getting closer every year.
- When partitioning/federating/sharding data, take the step to model what you are doing.
- Prototype several different schemes and make sure you truly understand your intended use patterns before deciding.

OmniTI

Networking

OmniTI / actually delivering

Techniques / *Networking*

- The network is part of the architecture.
- So often forgotten by the database engineers and the application coders and the front-end developers and the designers.
- Packets per second, firewall states, load balancing algorithms, etc.
- Many apps today are so poorly designed that network issues never become scalability concerns... others can really toss the bits.
- This is for the application architectures that have high traffic rates.

OmniTI

Networking / *basics*

- Scalability on the network side is all about:
  - understanding the bottleneck
  - avoiding the single point of failure
  - spreading out the load.

Networking / *going past gigE*

- A single machine can push 1 GigE.
- Actually more than a GigE isn't too hard.
- But how to push 10 or 20?
- Buy a really expensive load balancer?
- ... there are other ways to manage this a bit cheaper.

Networking / *going past gigE*

- use routing.
- routing supports extremely naive load balancing.
- run a routing protocol on the front-end ‘uber-caches’
- have the upstream use hashed routes
- the user-caches announce the same IP.
- this adds fault-tolerance and distributes network load.
- and it is pretty much free (no new equipment in the path).
- *note: your ‘uber-caches’ may be load balancers themselves.*

OmniTI

Networking / *isolation*

- for those that run multiple services on the same network.
- one service bursting on a.b.c.67 might saturate firewall and/or load-balancer capacity and degrade services other services behind the same infrastructure.
- again... routing to the rescue.
- set up a separate set of firewalls/load-balancers that reside in a “surge” net. Those firewalls only need to announce the /32 of the surging service to assume control of the traffic.

  note: you need some trickery to make sure return traffic is symmetric
- This is the same technique used to protect against DDoS attacks.

Service Decoupling

OmniTI / controlling experience by removing 'the suck'

Techniques / *Service Decoupling*

- One of the most overlooked techniques for building scalable systems
- Why do now what you can postpone until later?
  - This mantra often doesn't break a user's experience.
- Break down the user transaction into parts.
- Isolate those that could occur asynchronously.
- Queue the information needed to complete the task.
- Process the queues "behind the scenes."

Decoupling / *concept*

- If I don’t want to do something now…
- I must tell someone to do it later.

- This is “messaging”

- There are a lot of solutions:
  - JMS (Java message service)
  - Spread (extended virtual synchrony messaging bus)
  - AMQP (advanced message queueing protocol)

OmniTI

Decoupling / *tools*

- Message Queueing is the main tool used for this... durable message queueing:
  - ActiveMQ (Java)
  - OpenAMQ (C)
  - RabbitMQ (erlang)
- Most common protocol is STOMP
  - STOMP kinda sucks... but it is universal
  - Clients exist for every language

Decoupling / *tools*

- The typical use-case requires combining
  - a message queue, and
  - a job dispatcher
- People think Gearman does this (it does)
  - it does allow dispatching work across a cluster of machines
  - but, it doesn't inherently decouple the action from the outcome
  - yet, it is pretty straight forward to realize this
  - it can also be used to scale out work that *cannot* be decoupled.

OmniTI

Decoupling / *control*

“Moderation in all things, including moderation.”

- Titus Petronius
AD 27-66

WTF

WTF / *don't be an idiot*

- most acute scalability disasters are due to idiots
- don't be an idiot
- scaling is hard
- performance is easier
- extremely high-performance systems tend to be easier to scale
  - because they don't have to

SCALE

as much.

OmniTI

WTF / *sample 1*

- Hey! let's send a marketing campaign to:

  http://example.com/landing/page

-
  ```
  GET /landing/page HTTP/1.0
  Host: example.com

  HTTP/1.0 302 FOUND
  Location: /landing/page/
  ```

WTF / *sample 2*

- commit message: “prevent caching here.”

```
    swfobject.embedSWF(
-       "/XXXXX/swf/gallery.swf",
+       "/XXXXX/swf/gallery.swf?t=" + new Date().getTime(),
      "flashcontainer",
```

- caching should be ***controlled*** not prevented.

WTF / *sample 3*

- I have 100k rows in my users table...
- I'm going to have 10MM...
- I should split it into 100 buckets,
  with 1MM per bucket so I can scale to 100MM.
- The fundamental problem is that I don't *understand* my problem.

- I know what my problems are with 100k users... or do I?
- There is some margin for error...
  you design for 10x...
  as you actualize 10x growth you will (painfully) understand that margin.
- Designing for 100x let alone 1000x
  requires a ***profound*** understanding of their problem.
- Very few have that.

OmniTI

WTF / *sample 4*

- I plan to have a traffic spike from (link on MSN.com)
- I expect 3000 new visitors per second.
- My page http://example.com/coolstuff is 14k
  2 css files each at 4k
  1 js file at 23k
  17 images each at ~16k
  (everything's compressed)
- /coolstuff is CPU bound (for the sake of this argument)
  I've tuned to 8ms services times...
  8 core machines at 90% means 7200ms of CPU time/second...
  900 req/second per machine...
  3000 v/s / 900 r/s/machine / 70% goal at peak rounded up is...
  5 machines (6 allowing a failure)
- the other files I can serve faster... say 30k requests/second from my Varnish instances... 3000 v/s * 20 assets / 30k r/s/varnish / 70% is...
  3 machines (4 allowing a failure).

OmniTI

WTF / *sample 4, the forgotten part*

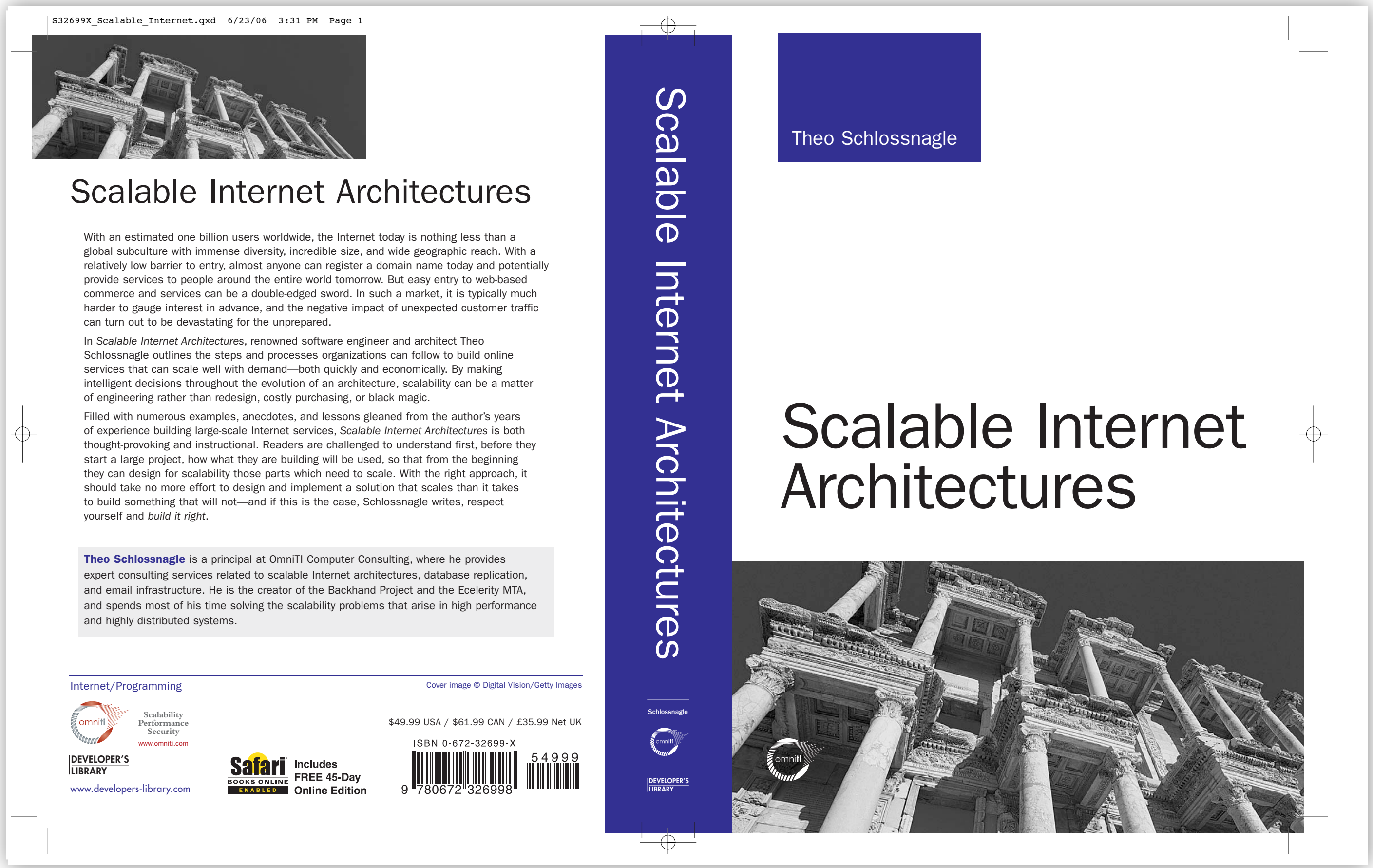
- 14k + 2 * 4k + 1 * 23k + 17 * 16k = 21 requests with 317k response
- (317k is 2596864 bits/visit) * 3000 visits/second = 7790592000 b/s
- just under 8 gigabits per second.
- even naively, this is 500 packets per visitor * 3000 visitors/second
- 1.5MM packets/second.

- This is no paltry task...
- 20 assets/visit are static content, we know how to solve that.
- the rest? ~350 megabits per second and ~75k packets/second
- perfectly manageable, right?
- a bad landing link that 302's adds ~30k packets/second... Crap.

OmniTI

Thank You

- Thank you Apache Software Foundation
  - 10 years. Wow! and How!
- Thank you OmniTI
  - We’re always looking for a few good engineers!
- Thank you!



S32699X_Scalable_Internet.qxd 6/23/06 3:31 PM Page 1

Scalable Internet Architectures

With an estimated one billion users worldwide, the Internet today is nothing less than a global subculture with immense diversity, incredible size, and wide geographic reach. With a relatively low barrier to entry, almost anyone can register a domain name today and potentially provide services to people around the entire world tomorrow. But easy entry to web-based commerce and services can be a double-edged sword. In such a market, it is typically much harder to gauge interest in advance, and the negative impact of unexpected customer traffic can turn out to be devastating for the unprepared.

In *Scalable Internet Architectures*, renowned software engineer and architect Theo Schlossnagle outlines the steps and processes organizations can follow to build online services that can scale well with demand—both quickly and economically. By making intelligent decisions throughout the evolution of an architecture, scalability can be a matter of engineering rather than redesign, costly purchasing, or black magic.

Filled with numerous examples, anecdotes, and lessons gleaned from the author’s years of experience building large-scale Internet services, *Scalable Internet Architectures* is both thought-provoking and instructional. Readers are challenged to understand first, before they start a large project, how what they are building will be used, so that from the beginning they can design for scalability those parts which need to scale. With the right approach, it should take no more effort to design and implement a solution that scales than it takes to build something that will not—and if this is the case, Schlossnagle writes, respect yourself and *build it right*.

Theo Schlossnagle is a principal at OmniTI Computer Consulting, where he provides expert consulting services related to scalable Internet architectures, database replication, and email infrastructure. He is the creator of the Backhand Project and the Ecelerity MTA, and spends most of his time solving the scalability problems that arise in high performance and highly distributed systems.

Internet/Programming

Cover image © Digital Vision/Getty Images

Scalability Performance Security
www.omniti.com

DEVELOPER’S LIBRARY
www.developers-library.com

Safari BOOKS ONLINE ENABLED Includes FREE 45-Day Online Edition

$49.99 USA / $61.99 CAN / £35.99 Net UK

ISBN 0-672-32699-X

9 780672 326998 54999

Scalable Internet Architectures

Schlossnagle

DEVELOPER'S LIBRARY

Theo Schlossnagle

Scalable Internet Architectures