

SCALABLE INTERNET ARCHITECTURES

GEORGE SCHLOSSNAGLE

THEO SCHLOSSNAGLE

<george@omniti.com>

<theo@omniti.com>



AGENDA

- ❑ CHOOSING HARDWARE
- ❑ CHOOSING AN APPLICATION/AVAILABILITY ARCHITECTURE
- ❑ DECIDING BETWEEN THIRD-PARTY AND CUSTOM-BUILT SOFTWARE
- ❑ CASE STUDY I: BUILDING FAST SCALABLE WEB FORUMS
- ❑ CASE STUDY II: DISTRIBUTED LOGGING



CHOOSING HARDWARE

- 'ENTERPRISE' HARDWARE
 - EXPENSIVE
 - RELIABLE
- COMMODITY HARDWARE
 - CHEAP
 - FAST
 - UNRELIABLE



SETTING UP APACHE

- TURN KEEPALIVES OFF
- AS CLUSTER GROWS, KEEP MAXCLIENTS TUNED TO AVOID EXCESSIVE DATABASE CONNECTIONS
- FOR DYNAMIC CONTENT CONSIDER USING A LOCAL PROXY INSTANCE



CONFIGURING A LOCAL PROXY

- RUN 2 APACHE INSTANCES ON A SINGLE HOST
- PUBLIC INSTANCE HANDLES HIGH-LATENCY CLIENTS USING MOD_REWRITE/MOD_PROXY.
- LOCAL INSTANCE HANDLES DYNAMIC CONTENT - ONLY MAKES LOW-LATENCY CONNECTIONS TO THE PUBLIC INSTANCE



CONFIGURING A LOCAL PROXY

- EXTERIOR (PROXY) INSTANCE

```
" <IfDefine PROXY
"
" DocumentRoot /var/apache/htdocs
"
" Listen myexternal_ip:80
"
" MaxSpareServers      32
" MaxClients           128
" MaxRequestsPerChild  100000
" KeepAlive            off
"
" LoadModule proxy_module libexec/libproxy.so
" LoadModule rewrite_module libexec/mod_rewrite.so
" AddModule mod_proxy.c
" AddModule mod_rewrite.c
"
" ProxyRequests        on
"
" NoCache
"
" ProxyPassReverse     / http://127.0.0.1
" RewriteRule ^proxy: - [F]
" RewriteRule ^(http:|ftp:) - [F]
" RewriteRule ^/(.*\.html)$ http://127.0.0.1/$1 [P,L,T]
" </IfDefine>
```



CONFIGURING A LOCAL PROXY

- INTERIOR INSTANCE

```
" <IfDefine DYNAMIC>
"
" DocumentRoot /var/apache/htdocs
"
" Listen                localhost:80
"
" MaxClients            40
"
" MaxRequestsPerChild  0
"
" KeepAlive             off
"
" LoadModule perl_module libexec/libperl.so
"
" AddModule mod_perl.c
"
" <Files *.asp>
"
" SetHandler perl-script
"
" </Files>
"
" </IfDefine>
```

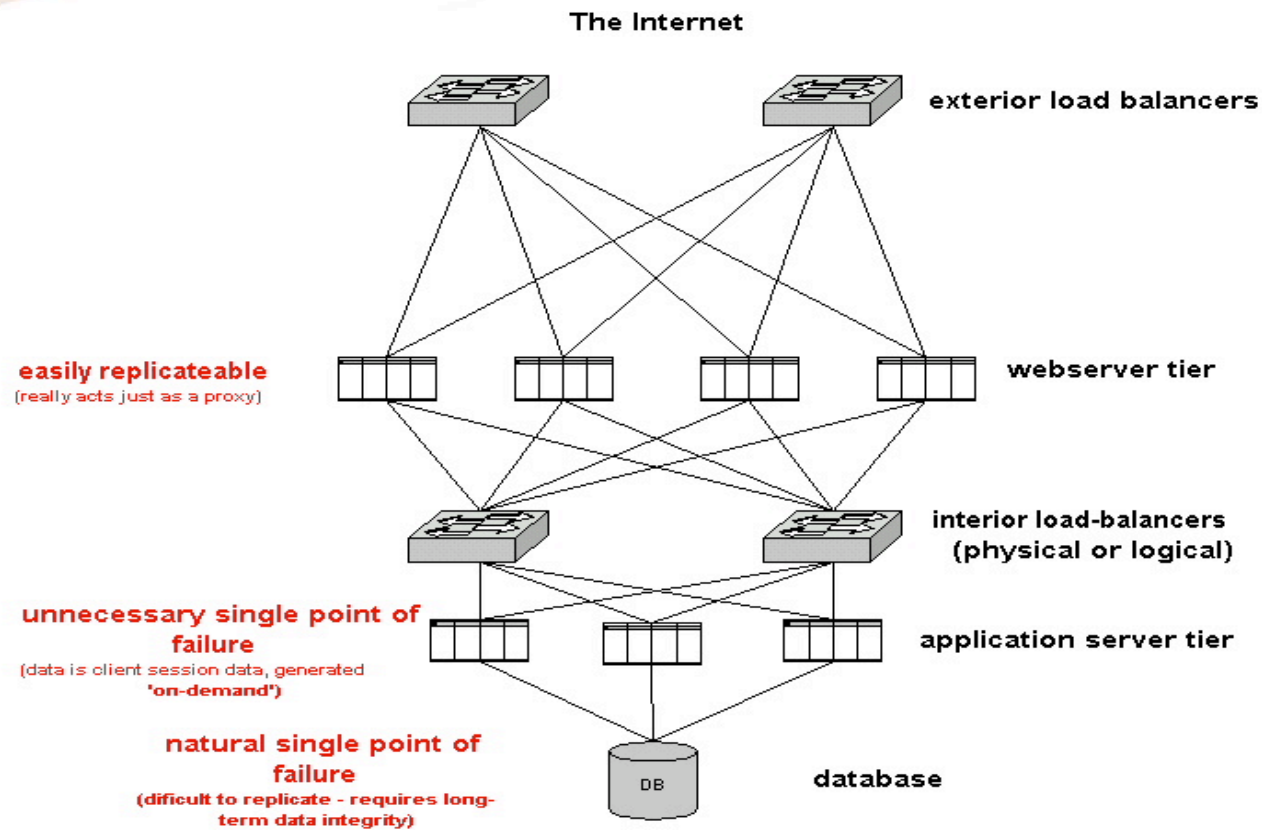


DESIGNING A HA/LB SCHEME THAT'S RIGHT FOR YOU

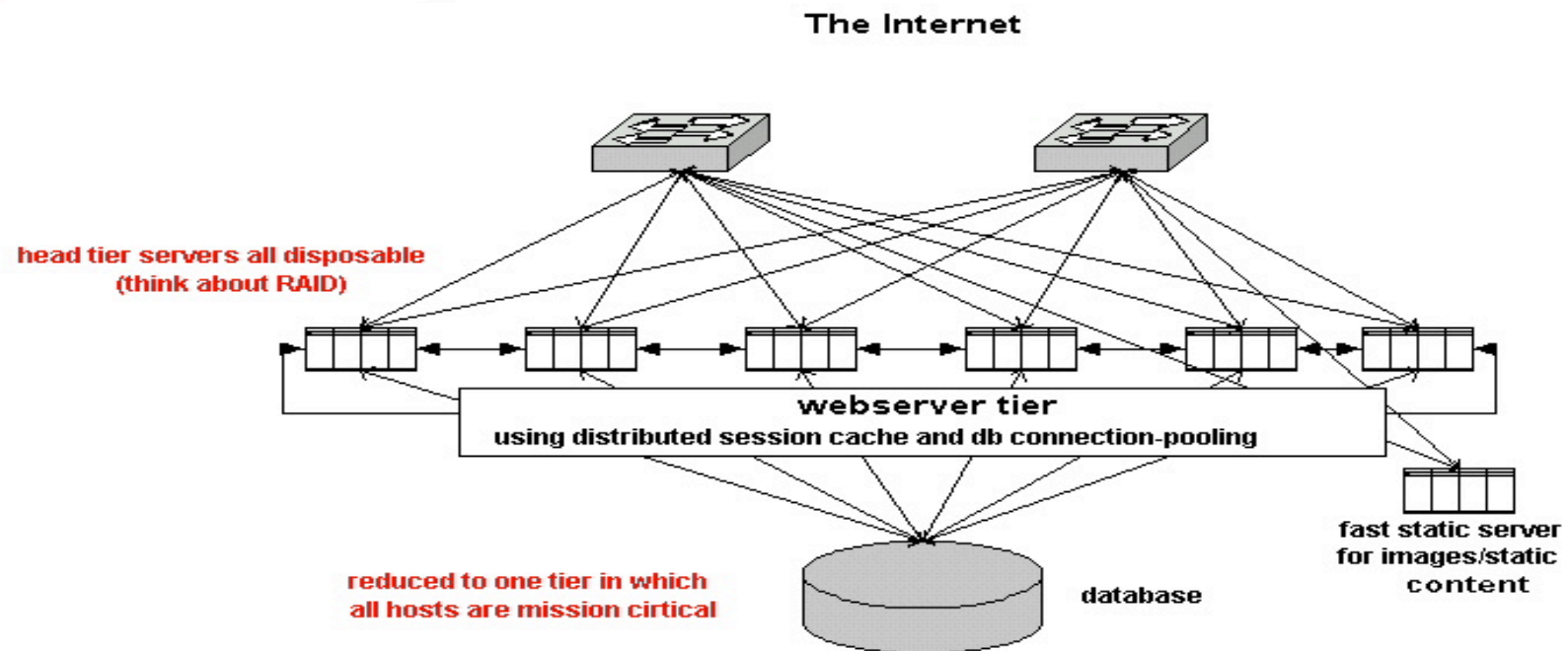
- RECOGNIZE THE DIFFERENCE BETWEEN REPLICATEABLE DATA AND NON-REPLICATEABLE DATA
 - REPLICATEABLE DATA NEEDS MARGINAL PROTECTION. USE COMMODITY HARDWARE.
 - NON-REPLICATEABLE DATA NEEDS SINGLE-POINT RELIABILITY, CONSIDER ENTERPRISE HARDWARE.
- BRING THE DATA TO THE SESSION, NOT VICE-VERSA.
 - LEVERAGE DISTRIBUTED SYSTEMS TECHNOLOGY
 - AVOID CREATING ARTIFICIAL POINTS OF FAILURE



TYPICAL THREE TIER ARCHITECTURE



MODERN TWO TIER MODEL



CHOOSING BETWEEN CUSTOM AND COMMERCIAL SOFTWARE

COMMERCIAL

- ✂ CODE 'MATURITY'
- ✂ DEDICATED SUPPORT

HOMEGROWN

- ✂ DESIGNED FOR YOUR PARTICULAR NEEDS
- ✂ IN-HOUSE SUPPORT



CASE STUDY I:

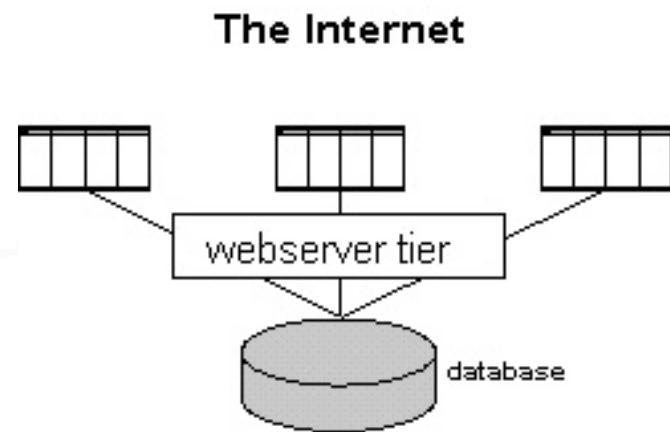
CACHING WEB OBJECTS

- HOW WELL DOES YOUR DATA MATCH THE ORIGINAL DESIGN GOALS OF ANY COMMERCIAL PRODUCTS BEING CONSIDERED?
 - IS THE DATA STATIC?
 - IS THE DATA STATIC FOR A SHORT PERIOD OF TIME?
 - IS THE DATA STATIC FOR A SHORT PERIOD OF TIME FOR EACH CLIENT?
 - DOES THE DATA CONTAIN COMPONENTS WHICH ARE STATIC FOR EACH CLIENT FOR A SHORT PERIOD OF TIME?



DETAILED EXAMPLE: WEB FORUMS ORIGINAL IMPLEMENTATION

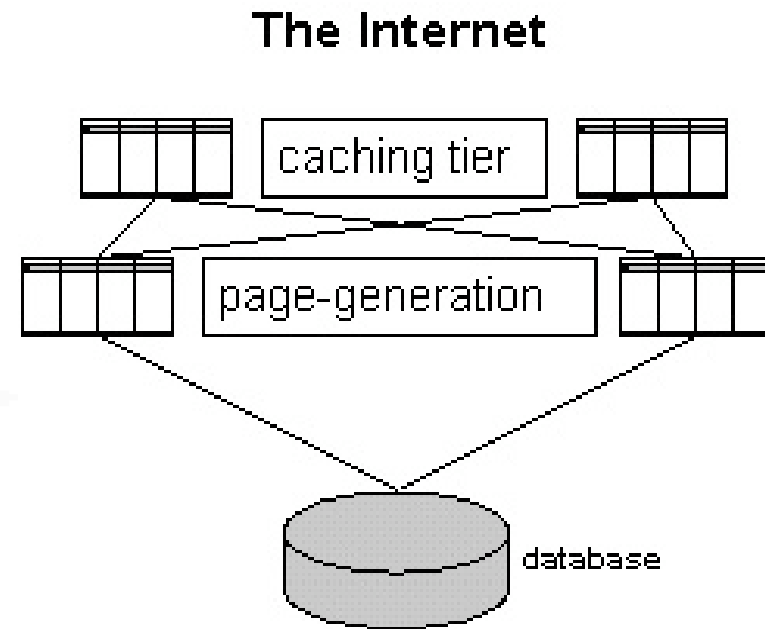
- ❑ EVERY PAGE IS GENERATED BY A DATABASE QUERY WHICH RETURNS A SORTED LIST OF ALL MESSAGES WHICH ARE RETURNED TO THE USER.
- ✂ INEFFICIENT, DATABASE INTENSIVE, SCALES POORLY AS MESSAGE VOLUME INCREASES.
- ✂ TAKES NO ADVANTAGE OF SELECT/UPDATE RATIO.



SECOND IMPLEMENTATION: ADD BLACK-BOX CACHING

LAST-MODIFICATION TIME IS STORED ON EVERY UPDATE AND IS USED TO MARK MESSAGE LISTINGS AS CACHEABLE.

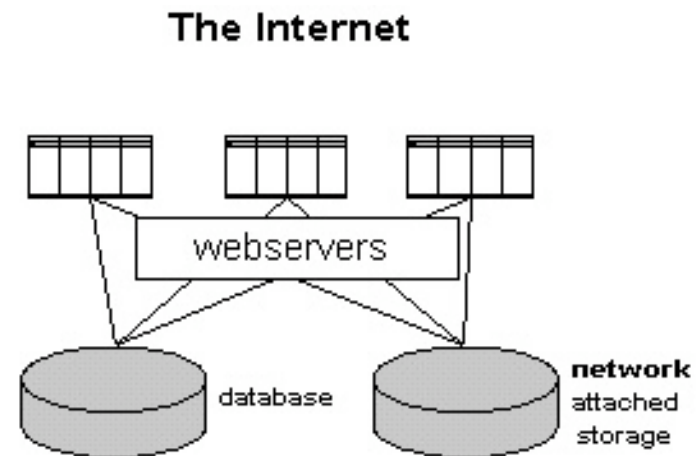
- q TAKES ADVANTAGE OF HIGH CACHE LOCALITY.
- q PROVIDES GOOD SCALEABILITY RESULTS.
- q REQUIRE 3-TIER ARCHIECTURE.
- q MINIMAL APPLICATION MODIFICATION REQUIRED.



THIRD IMPLEMENTATION: APPLICATION-INTEGRATED CACHING

STATIC PAGES ARE WRITTEN TO SHARED FILESYSTEM, AND
REWRITTEN ON UPDATE

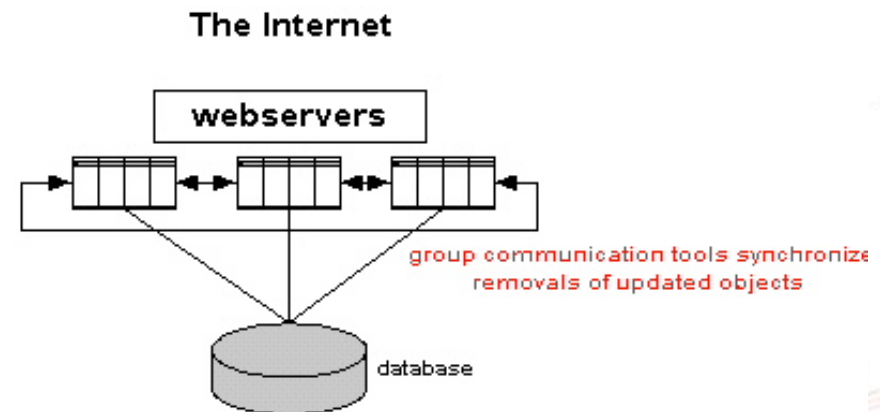
- ✂ TAKES ADVANTAGE OF HIGH
CACHE LOCALITY.
- ✂ EFFICIENT USE OF
HARDWARE.
- ✂ GOOD SCALABILITY



FOURTH IMPLEMENTATION: APPLICATION-INTEGRATED CACHING (II) LEVERAGING DISTRIBUTED SYSTEMS TECH

STATIC PAGES ARE WRITTEN LOCALLY, NODES USE GROUP COMMUNICATION TOOLS TO COORDINATE STATIC PAGE REMOVAL ON UPDATES.

- q IDEAL USE OF COMMODITY HARDWARE.
- q TAKES ADVANTAGE OF HIGH CACHE LOCALITY.
- q EXCELLENT SCALEABILITY AND AVOIDANCE OF SPOFs.



IMPLEMENTATION

- MOD_REWRITE SETUP

```
RewriteCond %{REQUEST_FILENAME} !-f
```

```
RewriteRule ^/forums/(.*)$ /admin/generator.php?forumid=$1
```



IMPLEMENTATION

- GENERATOR.PHP

```
" <?php
"
"     $forumid = $_GET['forumid'];
"
"     if(!$uri) {
"         return_error();
"     }
"     ob_start();
"     if(generate_page($forumid)) {
"         $content = ob_get_contents();
"         $fp = fopen($SERVER['DOCUMENT_ROOT'].$uri, "w");
"         fwrite($fp, $content);
"         ob_flush();
"     }
"     ob_clean();
"     return_error();
" ?>
```



IMPLEMENTATION

- UPDATE PAGE:

- <?php

- ...

- update_page(\$uri);

- purge_cache(\$uri);

- ?>

- purge_cache CAN BE SOMETHING AS SIMPLE AS `unlink()` IF WE HAVE A SINGLE MACHINE OR ARE USING A SHARED MOUNTPOINT. OTHERWISE WE CAN USE SOMETHING LIKE SPREAD TO COORDINATE POISONING OF ALL THE CACHES.



CASE STUDY II: DISTRIBUTED LOGGING

- NEED TO CONSOLIDATE LOGS ACROSS MULTIPLE WEBSERVERS FOR AUDITING
- NEED TO DO REAL-TIME ANALYSIS OF LOGS



FIRST (TRADITIONAL) IMPLEMENTATION

- WEB LOGS WRITTEN LOCALLY ON EVERY MACHINE, PERIODICALLY COPIED TO CENTRAL SERVER AND SORTED/MERGED
 - CONSOLIDATION IS SLOW
 - REAL TIME LOG PROCESSING IS NOT POSSIBLE



Candidate Solutions

COMMERCIAL SOLUTIONS

- EXPENSIVE
- LACK FLEXIBILITY

EXISTING OPEN SOURCE SOLUTIONS

- SYSLOG LOGGING
 - UNRELIABLE
 - UNICAST
- DATABASE LOGGING
 - RELIABLE
 - UNICAST



CUSTOM SOLUTION (MOD_LOG_SPREAD)

- DESIGNED AS APACHE MODULE FOR MAINTAINABILITY
- RELIABLE MULTICAST TRANSPORT FOR MAXIMUM FLEXIBILITY
- AGGREGATED LOG STREAM CAN BE USED TO MAINTAIN/TRACK USER STATE AND SERVER HEALTH ACROSS MULTIPLE SERVERS, ASYNCHRONOUSLY BUT IN REAL-TIME
- MULTICAST TRANSPORT ALLOWS ADDITIONAL MONITORING FACILITIES TO BE ADDED FOR 'FREE'



THANKS!

