Scalable Hosting of Web Applications

Guillaume Pierre
(with Zhou Wei, Jiang Dejun, Swaminathan Sivasubramanian, Tobias Groothuyse, Sandjai Bhulai, Chi-Hung Chi and Maarten van Steen)

CANOE and EuroSys Summer School

21 august 2009

http://www.cs.vu.nl/~gpierre/

vrije Universiteit amsterdam

Advertisement

- This school is co-organized by EuroSys
  - The European Professional Society on Computer Systems
  - Scope: operating systems, distributed systems, event-based systems, embedded systems, etc.
  - Membership: 40 euros (senior), 10 euros (students)
- Upcoming activities:
  - EuroSys VMware Premier Conference Award (application deadline: August 28th)
  - EuroSys Shadow PC (application deadline: September 15th)
  - EuroSys 2010 conference (submission deadline: October 23rd)
  - Roger Needham PhD award (application deadline: December 12th)
  - Note: it is not necessary to be a member to participate!

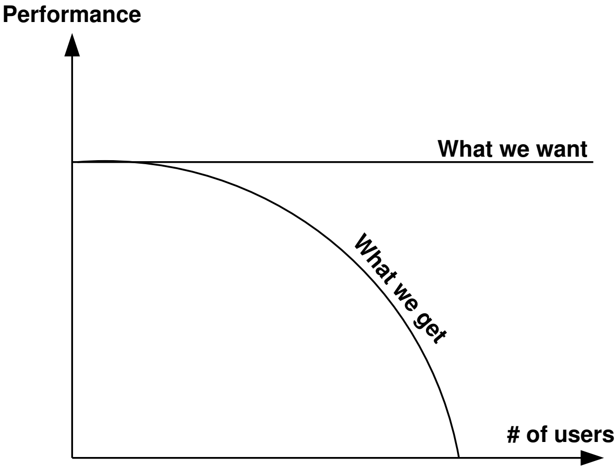
www.eurosys.org

The Problem

1. You build a great Web site, advertise it
2. . . .

The Problem

1. You build a great Web site, advertise it
2. . . .



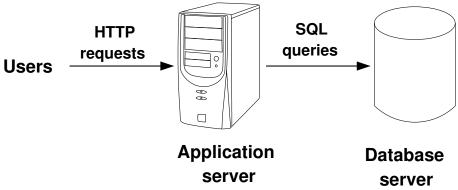
Scalability

"A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity."

B. Clifford Neuman,
"Scale in Distributed Systems"

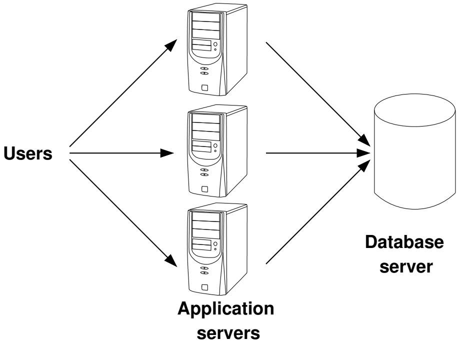
A typical Web application

- One application server runs application code
- One database server holds the application state
- The code can issue any query to the database
  - SELECT (read queries)
  - UPDATE, DELETE, INSERT (UDI queries)
  - Transactions



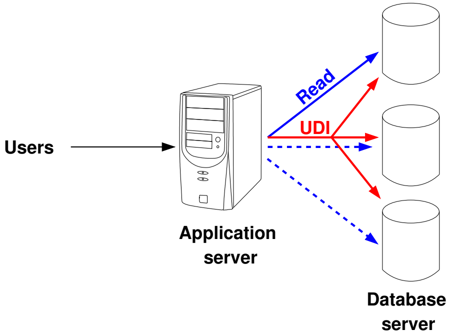
Scaling the application server

- The application server contains only the application code
  - It does not hold state
  - Different requests can be processed independently



Replicating the database server

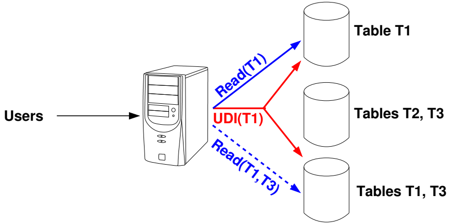
- State is fully replicated across multiple database servers
  - Read queries can be addressed at any replica
  - UDIs must be issued at every replica



- Each database server must process $\frac{1}{N} Read\_Queries + UDIs$ query load
  - Increasing N does not help when the UDIs alone saturate the server's capacity

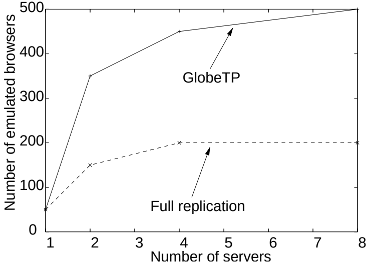
Partially replicate the database

- We must send less UDIs to each server
  - Let's partition the database
  - Each server contains a subset of all tables

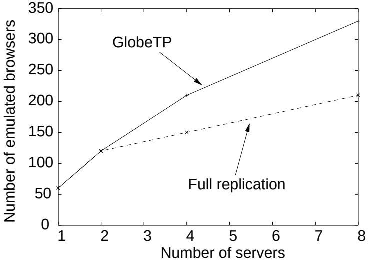


  - Updates to T1 must be addressed to only 2 servers
  - We must place tables according to query templates
    - ★ We cannot execute a query that joins T1 and T2...

Performance of partial database replication



TPC-W (e-commerce app)



RUBBoS (Slashdot-like)

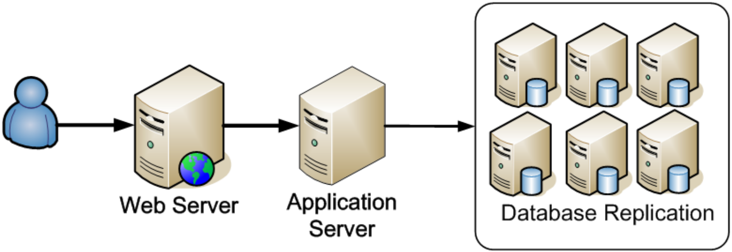
- Problem: table-level granularity is too coarse
  - Maximum gain = # of tables
  - We need a finer granularity: column-level

Table of Contents

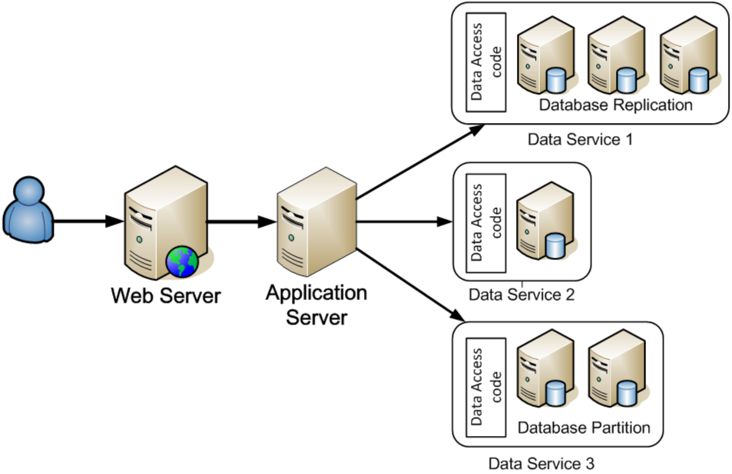
Position

- We must split the application data into a number of independent services
  - This implies restructuring the data schema at the column granularity
- Each data services has its own private data store
  - It can be accessed through a well-defined interface
- This transformation does not improve performance!
  - But it makes the workload of each service much simpler
  - It is easier to scale each service independently

System model (traditional)



System model (denormalized)

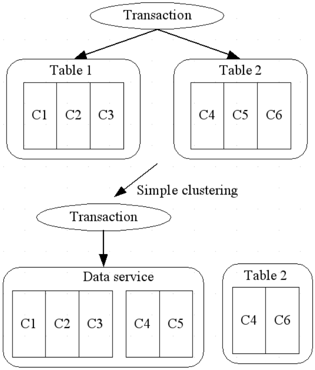


Can we split data arbitrarily?

- Answer: of course not!
  - Queries and transactions access multiple data rows simultaneously
  - We must make sure that the application queries can still execute
  - Pay particular attention to transactional ACID properties
- We must restructure the data according to the queries and transactions

Step 1: restructure data according to transactions

- A transaction may access any number of data items
  - For consistency these items must remain inside the same data service
  - Let's cluster data items according to transaction patterns



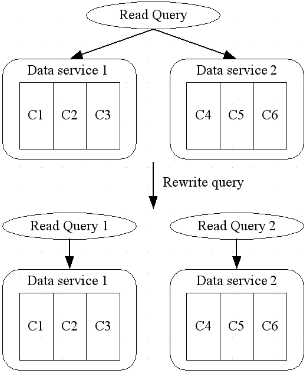
Step 2: restructure data according to regular queries

- Problem: many queries may now access data from multiple data services
  - Naive solution: cluster data services according to regular queries
  - But this would result into a single monolithic cluster
- Instead, we can apply other transformations
  - Rewrite complex queries into multiple simple queries
  - Replicate read-only columns across multiple data services
  - In last resort, merge data services

Rewrite complex queries

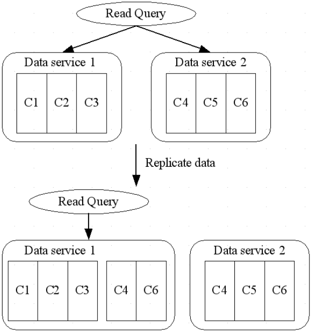
- Many join queries can be rewritten into several simple queries
- Example: `SELECT C6 FROM T1,T2 WHERE T1.C1 = ? AND T1.C2 = T2.C5`
- This query can be rewritten into:
  1. `SELECT C2 FROM T1 WHERE T1.C1 = ?`
  2. `SELECT C6 FROM T2 WHERE T2.C5 = ?`

  The result of query 1 is the imput of query 2



Replicate read-only column

- Original query: `SELECT T1.C1, T1.C2 FROM T1,T2 WHERE T1.C1 = T2.C4 AND T2.C6 = ?`
- Columns `T2.C4` and `T2.C6` are read-only in the whole application
  - We can replicate them across multiple data services



Scaling each data service

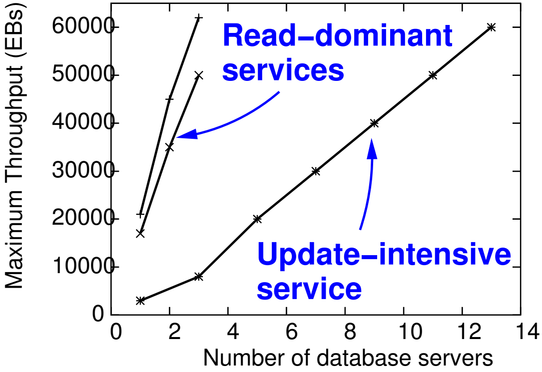
- We studied the case of TPC-W
  - A standard benchmark modeling an e-commerce site
  - Standardized workload
- Before denormalization:
  - 10 tables, 6 transactions, 2 atomic sets, 6 UDI queries that are not part of a transaction, and 27 read-only queries
- After denormalization:
  - 8 data services, in total 15 tables
- Important observation: most data services are read-dominant
  - Database replication works well for them
- Only one data service is update-intensive
  - Database replication will not work here, we need to pay closer attention

Scaling the Financial service

- The update-intensive service contains all financial-related operations
  - Shopping carts, orders, item stocks
- Most queries are index by shopping cart ID
- We can apply horizontal partitioning:
  - Hash table records by their shopping cart ID
  - Place each record on a different server according to the hash
  - Consequence: UDIs must be addressed to only one server

Performance of individual data services

- We define a response time objective: 90% of service invocations must return in less than 100 ms
- When using N servers, how many simultaneous clients can we support before violating the objective?



Performance of the entire application

- Response time objective: 90% of client requests must return in less than 500 ms

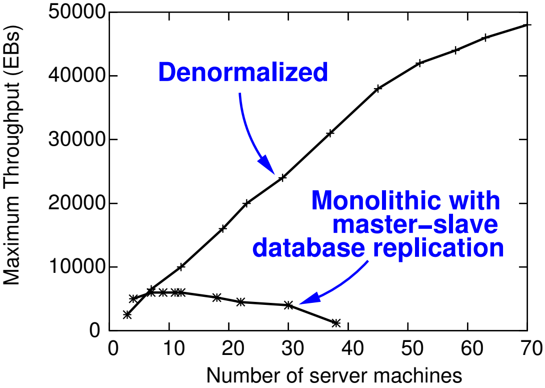
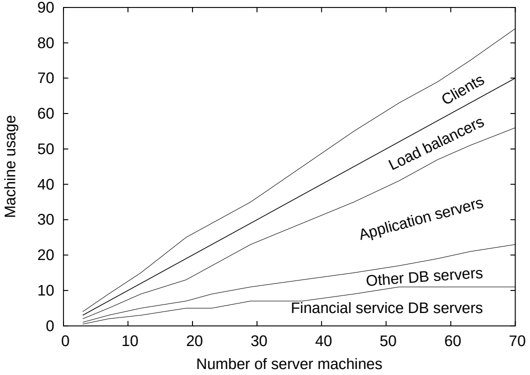


Table of Contents

The "secret sauce" behind the previous graph

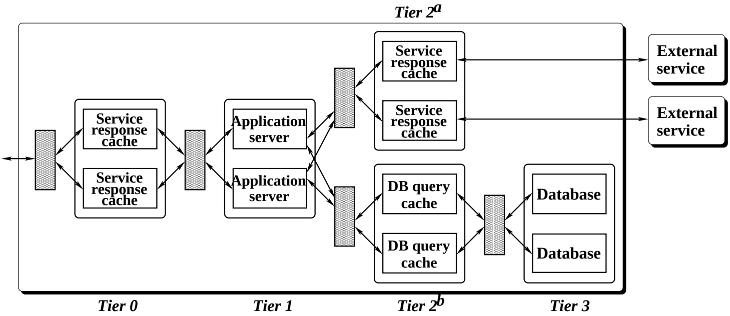
- How did we plot the previous graph?
  - For each configuration we must select what each machine will do



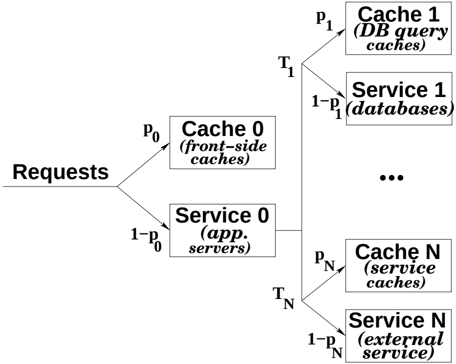
- Method: trial and error :-(
  - This is not acceptable in a real Web hosting environment...

Resource provisioning for a single Web service

- One Web service can be seen as being composed of:
  - 0 or more front-side cache(s)
  - 1 or more application server(s)
  - 0 or more database query cache(s)
  - 0 or more database server(s)
  - 0 or more external service response cache(s)



We can model a Web service as a queuing network



- Model:
  - Poisson distribution of arrival times
  - Infinite-server queue caches
  - Processor-sharing application servers and database servers

Mean response time

- We can calculate the mean response time:

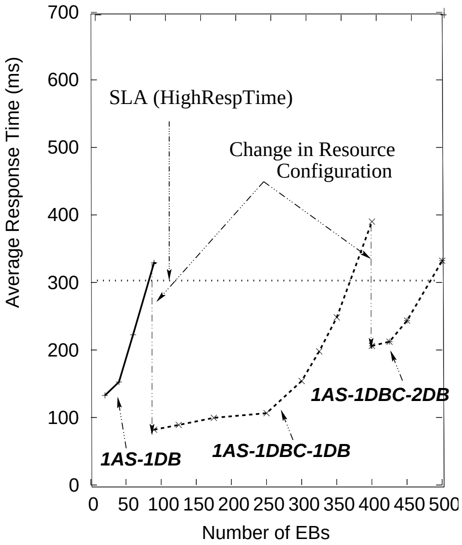
$$\mathbb{E}S = p_0\beta_{c,0} + (1-p_0)\frac{(M+1)\beta_{s,0}}{1-\rho_{s,0}} + (1-p_0)\sum_{i=1}^{N}\mathbb{E}T_i\left[p_i\beta_{c,i} + (1-p_i)\frac{\beta_{s,i}}{1-\rho_{s,i}}\right].$$

- The formula for the variance looks much worse. . .

Model-based resource provisioning

- The performance model allows us to steer resource provisioning
  1. Give an SLA to the service
  2. Monitor its response time
  3. When the SLA is violated: for each tier, compute the expected response time if this tier would have one more server
  4. Select the tier that brings the most improvement, add a server there
- Similar algorithm for removing servers when traffic decreases
- Note: there are a few subtleties
  - How do you estimate the new cache hit rate if you add more caches? (add more caches $\equiv$ increase cache size)
  - When should you initiate this process?

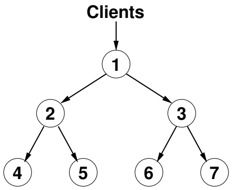
Example: TPC-App



Resource Provisioning of a multi-service application

- Nowadays most service-oriented applications use a graph of services
  - *"If you hit the Amazon.com gateway page, the application calls more than 100 services to collect data and construct the page for you."* [Werner Vogels, Amazon CTO]

- Simple option: **give an SLA to each service**
  - Service 1 has the same SLA as the whole application
  - How do you select SLAs for the other services?
  - A wrong choice leads to inefficient resource usage



- Our option: **give an SLA only to the front-side service**
  - Let services negotiate resource allocation with each other
  - "How much faster/slower can your sub-tree perform with one more/less machine?"

Example: a 7-service invocation tree

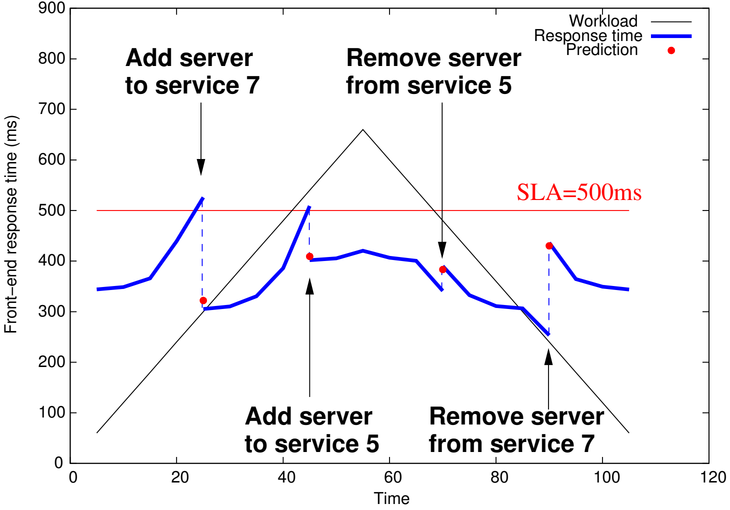


Table of Contents

Conclusion

- Web applications are very diverse
  - Most of them can easily be hosted by a single PC
  - Some of them require complicated infrastructures with thousands of servers
  - It is impossible to predict if a small site will become popular tomorrow!
- Even small Web applications should be ready to scale if necessary:
  1. Denormalize the application's data into independent services
  2. Enable hosting infrastructures with automatic resource provisioning mechanisms
  3. We need pools of resources that can be automatically assigned to applications (Grids, Clouds. . .)