CSC2231: Lessons from Giant-Scale Services

http://www.cs.toronto.edu/~stefan/courses/csc2231/05au

Stefan Saroiu

Department of Computer Science

University of Toronto

E-mail list

1. **Jing Su**
2. **Jin Chen**
3. **Jesse Pool**
4. **Ian Sin**
5. **Guoli Li**
6. **Madalin Mihailescu**
7. **Alex Wun**
8. **Nilton Bila**
9. **Kenneth Po**
10. **Troy Ronda**
11. **Hadi Bannazadeh**

} Take class for credit!

………..

1. **E-mail me to have your name here** } Audit/Attend lectures

Context

- **1994-1996: Web takes off**
  - 1993: Mosaic is released
    - Backbone Xfic starts growing 30x / year
  - 1994: Yahoo, Rolling Stones, Pizza Hut come to the Web
    - Commercialization of the Internet is starting
    - Scalability and availability are becoming important
  - 1995: Lycos, Inktomi, AltaVista commercially founded
    - Inktomi uses clusters for scalability and fault tolerance
    - AltaVista uses large-scale SMPs

Internet services: old problems, new domain

- **Scalability:**
  - Absolute: must serve large populations and high request rate
  - Incremental: grow system without throwing old system out
- **Availability:**
  - Downtime directly translates to lost $$$
    - 1 hour of downtime for financial e-commerce = US$ 6 mill.
  - Bound by availability of Internet itself
- **Cost-effectiveness:**
  - Hardware must be cheap and not wasted
  - Human costs start dominate hardware costs (manageability)

Internet services: new domain, old mistakes

- **Huge demand for cluster architectures that are:**
  - Scalable, available, cheap
- **Nobody worried too much about:**
  - Security
    - Internet was still perceived as a friendly environment
  - Internet properties
    - Availability of routing layer
    - Quality of service provisioning

New attributes of Internet domain

- **Different consistency semantics**
  - Web trains users to expect occasional, visible glitches
    - Reload consistency
      - Google queries are neither complete nor consistent
    - OK, as long as the system does not remain divergent
  - Use RDBMS for protecting data involving $$$
- **Embarrassingly parallel workloads**
  - Tasks are read-only and independent (see Google)
- **Graceful degradation makes sense**
  - Not all users or operations are equal
  - Partial data is still useful

Cluster computing: +’s and -’s

- **Clusters fit well many challenges of Internet services**
  - Scalability: embarrassingly parallel workloads
  - Availability: failure unit == cluster node
    - Have software provide fault-tolerance
  - Price/performance: use commodity nodes
- **New challenges:**
  - Manageability, administration (human costs >> hw costs)
  - Availability and performance in face of partial failures
  - No shared state between nodes
    - Maintaining state (write workloads) becomes v. hard

General research challenges

- **Build an Internet service toolkit for clusters**
  - Storage: parallel DB, distributed FS
  - Scheduling: load balancing switches, cost/affinity scheduling
  - Fault tolerance: failure detection, failover techniques
  - Recurring theme: exploit weaker semantics to simplify SW
- **Design patters for Internet services:**
  - Three-tier model: FE, middleware, DB back-end
- **Simplify administration:**
  - Eliminate human from the loop:
    - Functional homogeneity, automatic load balancing

Availability Metrics

- **Brewer argues for 2 metrics of "query-oriented" services**
  - Yield: fraction of queries that complete
  - Harvest: fraction of database captured in response

Service capacity:= Data/Query X Query/Sec

(Harvest) (Yield)

Handling Failures

- **Two ways:**
  - Partition: maintain yield, at the cost of harvest
  - Replication: maintain harvest, decreasing yield
    - Works great for read-only workloads
    - Data updates are hairy

Provision for Performance

- **System may be underprovisioned because of bursts**
  - If burst is short relative to user expectations

Provision for Performance

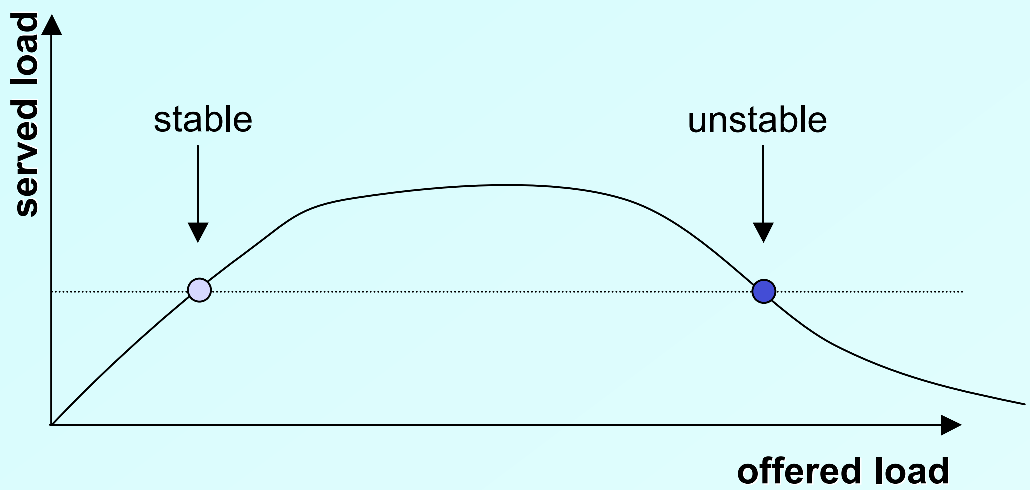
- **System may be underprovisioned because of bursts**
  - If burst is short relative to user expectations
    - Buffer (Web: seconds; P2P: hours)
  - If burst is chronic:

Provision for Performance

- **System may be underprovisioned because of bursts**
  - If burst is short relative to user expectations
    - Buffer (Web: seconds; P2P: hours)
  - If burst is chronic:
    - Over-provision
    - Admission control to degrade gracefully
    - App-specific ways to reduce harvest and preserve yield
      - e.g., drop expensive requests

Capacity

- **Overload shape:**



Classic Availability Metrics

- **Availability = (MTBF-MTTR) / MTBF**
  - Let's look at an example
  - Not all seconds have equal value
- **Claim: reducing MTTR is better than increasing MTBF**

Classic Availability Metrics

- **Availability = (MTBF-MTTR) / MTBF**
  - Let's look at an example
  - Not all seconds have equal value
- **Claim: reducing MTTR is better than increasing MTBF**
  - MTTR proportional to impact on user
  - MTTR proportional to cost to service
    - Tolerance threshold: low enough MTTR
  - MTTR is measurable, MTBF may not be

Discussion

- **Does rent-a-cluster makes sense?**

Discussion

- **Does rent-a-cluster makes sense?**
  - Virtual vs. physical clusters?
    - Build virtual clusters from virtual machines?
      - Cluster migration?

Discussion

- **Should Amazon.com have a massive cluster or should they have O(100s) of geographically displaced clusters?**

Discussion

- **Should Amazon.com have a massive cluster or should they have O(100s) of geographically displaced clusters?**
  - Scale vertically or scale horizontally?
    - Horizontal is cheaper and more fault-tolerant
    - But…, load balancing and failover are tricky