Scalable, Stable, Open

Long-term Goals for System Design

Rent the Runway in 2013

... Million users have created accounts

Concurrent user spikes up to...

A slow day sees traffic lows ...

More engaging site with more features

People stay longer and hit our servers more

Much more data to crunch in real time

Scalability

Horizontal: Throw more threads/processes/machines at it

Vertical: Throw bigger machines at it

Smart design uses both wisely

Horizontal Scalability

Any place you can shard big data or compute without needing crosstalk

- Web tier, sharded to handle more sessions
- Stateless services*
- User data*

Enables you to scale in near real-time by spinning up a new server and adding it to the pool

Vertical Scalability

Moore's law to the rescue!

When you need all the data in the same place to make a decision

When you need more processing power or memory per request

Mostly applies to the database

Used judiciously, this will enable us to scale with a lot less complexity

Generally requires some planning and downtime esp for DB

Code for Scalability

Never assume statefulness in a service connection

Don't cache write data unless you are the only writer

Cache read-only data as much as possible

Think about sharding points

Think about where you must have transactions

What can be done offline as aggregation vs required for real-time

Load Testing and Monitoring

Hard to do ad-hoc on non-production HW

Stage should be as close to prod as possible

- Memcached on and operational
- LB backend services

Automated performance testing as part of regression

- Compare time on same HW setup
- Performance regressions as serious as correctness regressions

Stability

As we scale, more moving parts to break

To keep things stable, who ya gonna call?

Open

Slowly but surely, we're opening the site

By EOY, it should be fully open

Don't assume a UserID is available

When do you need to force a user to log in?

Do you hide behaviors that require a logged in user? Which ones?

How do we gather good analytics and A/B test against logged-out users?

Systems, Now and Future

Data-tier services

- Reservations
- Product Catalog
- User Services (Ratings, Reviews, Events)
- Financials

These services own a type of data, and computation related only to that data

They do not work with data they don't own

They are the system of record for that data

Systems, Now and Future

Q: What do you do when you need data from two or more places?

A: Talk to Big Ben!

Big Ben is our integration service

Owns no data

Talks to other services and aggregates their results

Size runs: Product catalog sizes plus res cal stock availability

Outfits: Product catalog pairings plus res cal availabilty

Filtering: PC, ResCal, and User Ratings for hearted items

Systems, Now and Future

Data storage systems

MySQL of course, for all transactional data

MongoDB, for product metadata and some user data

Memcached for pure caching

Solr runs full-text search

Redis will probably be added for session caching (reads + writes)

HDFS/HBase possibly for analytics/stats log

Systems, Now and Future

Communication Layer

RESTful Java servers (Play!, Dropwizard, Glassfish) talking JSON

Embedded client speaking XML

JMS for transactional messaging and asynchronous processing

Load balancer to the backend services (possibly moving to HAProxy)

Final Thoughts

Always Assume Distributed
Plan for Open
Plan for Mobile
Don't write what you don't own
Cache it!
The future of tech at RtR is AWESOME