RFT Testing Clouds

Michael Freeman
Applications Development Analyst
Lender Processing Services
mlfreeman@gmail.com

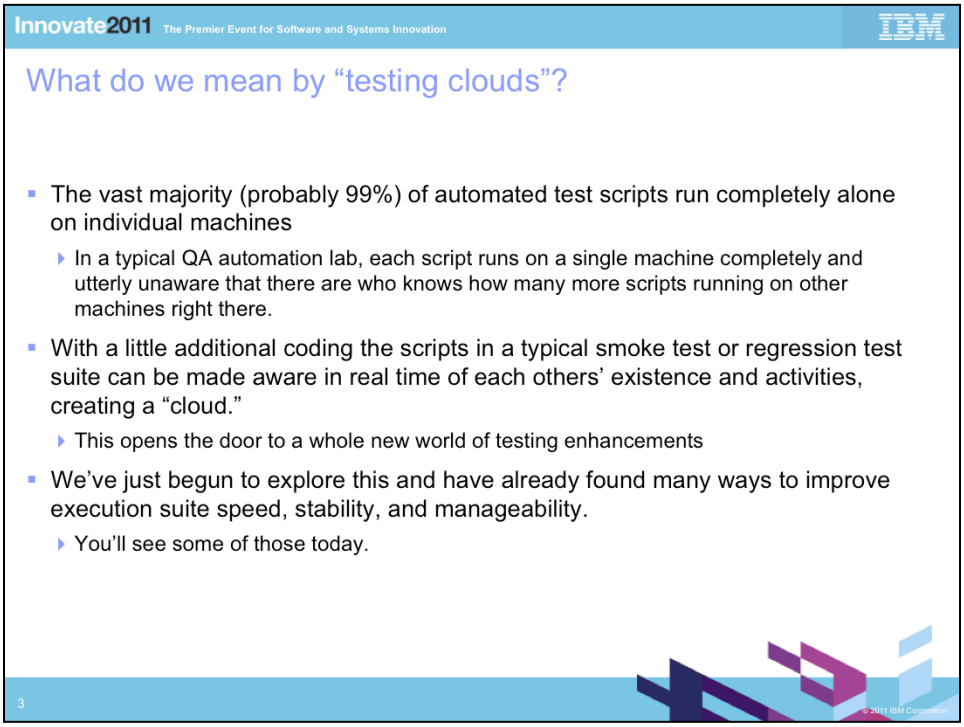
Latisha Aguilar
Applications Development Manager
Lender Processing Services
lzaguilar@gmail.com

QM-1284

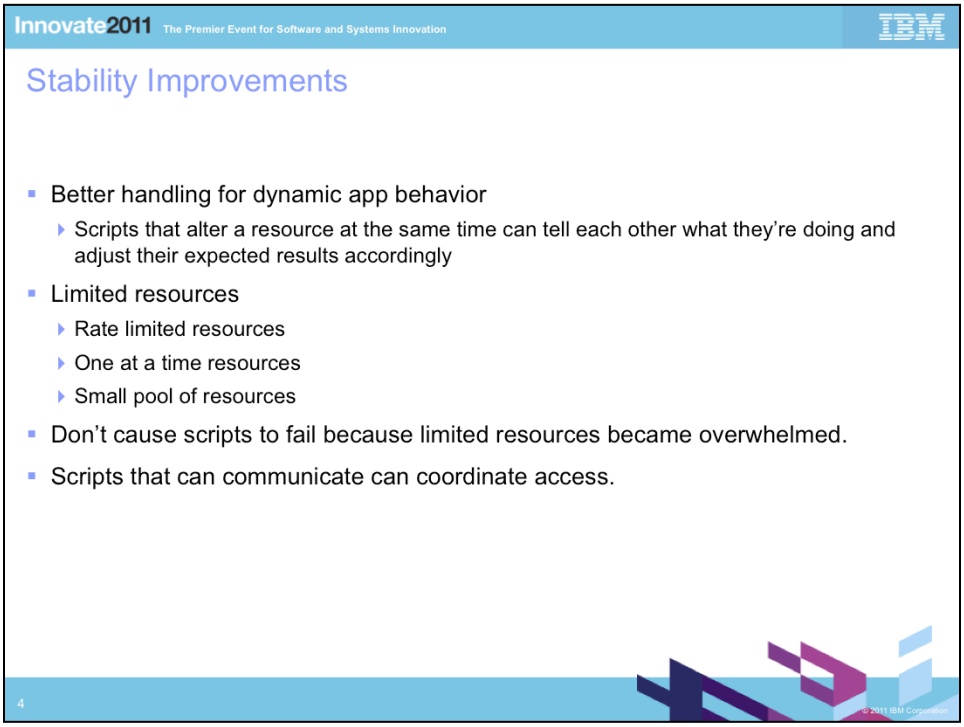
IBM Software
Innovate2011
The Premier Event for Software and Systems Innovation
Software. Everyware.
June 5–9 Orlando, Florida

Agenda

- Why connect RFT scripts?
  - What can you gain?
- When is it appropriate?
- How to connect RFT scripts
  - Hazelcast
  - Others
- Example scenarios
  - Where we've put this knowledge to use
- Demo
- Questions



You may have seen that the title of this slide set was “RFT Testing Clouds”. What do we mean by the term “Testing Clouds” ?



Example of "One at a time resources"

- LPS Desktop PM – one-session-per-user optional limit
- LPS Desktop IM – a vendor can submit only one invoice at a time per loan
  - Each loan in Invoice Mgmt can have many invoices from a vendor on it, but only one from that vendor can be in a "submitted" state at any moment

Example of "Small pool of resources"

- LPS Desktop IM <-> Fannie Mae integration enabled loans
  - Fannie Mae gave us only a small number of loan numbers to test with
  - We can't add more
  - There were more test scenarios than loan numbers
  - Some loan numbers would NOT work with certain test scenarios

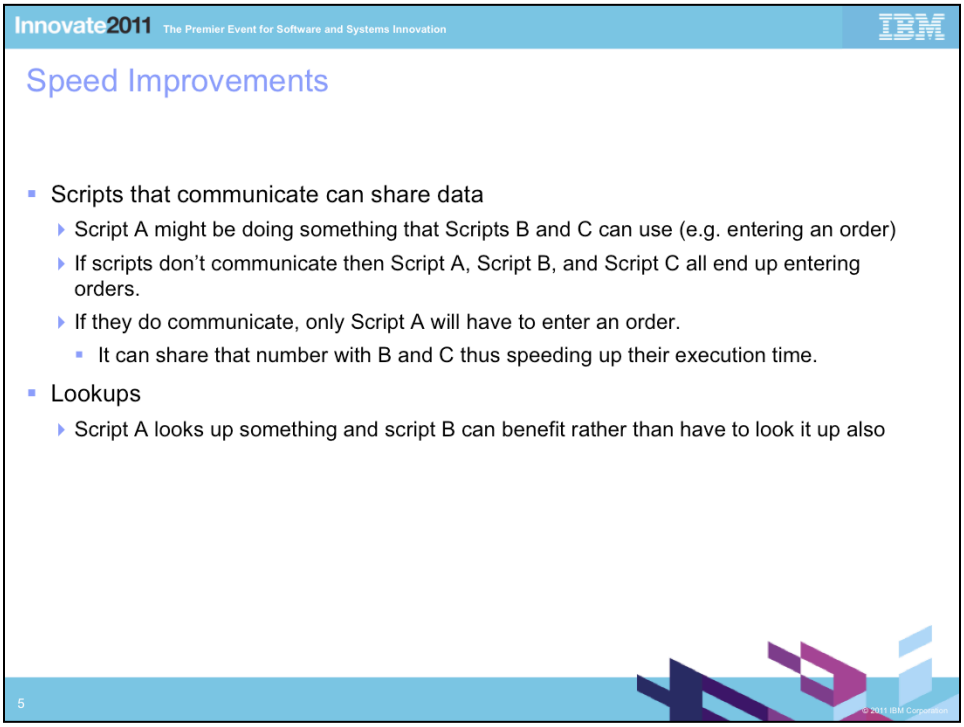
LPS Desktop - My Account

All our test users are shared

In some places we have to select our user using the user login

In some places we have to select our user using the first and last name

We need to be able to have scripts notify each other when the ones testing the "My



This was what started the clustering / communication thing for us.

We would launch scripts on our lab and we noticed that the first thing they ALL did was to enter orders into LPS Desktop PM…so we started wondering "can't a few of them create orders and the rest reuse them?"

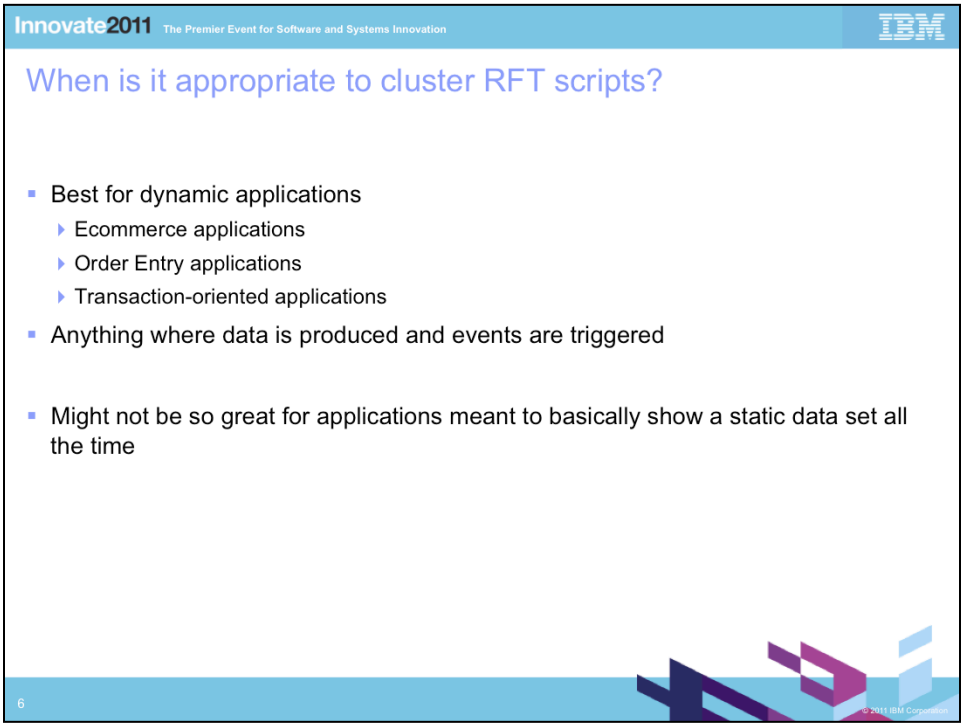
We couldn't stage data for several reasons

1. We run in multiple environments, and any of them could be refreshed or reset at almost any time (especially the developer environment)
2. Very easy for other users (e.g. manual QA team) to wind up altering pre-staged data
3. Configuration settings and other things can change between staging and running

- Sometimes automating the configuration steps is an acceptable workaround. Even better if you're lucky enough to have write access to your pre-prod environments (we were not so lucky) a few update statements can easily take care of the problem without opening your script up to more UI vulnerabilities.

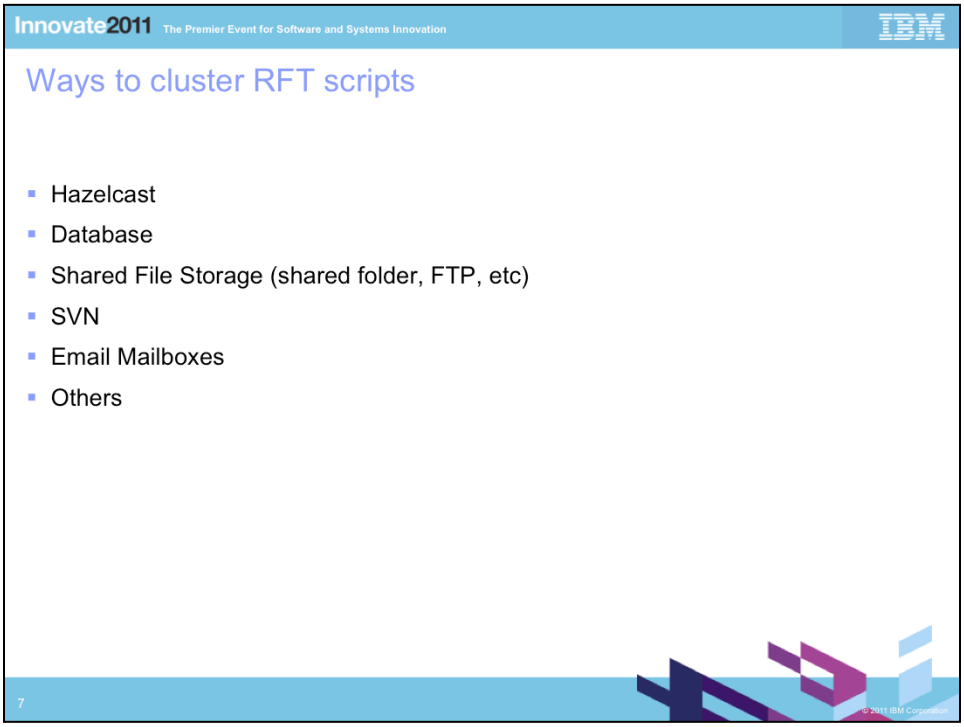
Calling into a library totally under your control is more stable than having to find objects, click, type, etc your way through the UI to grab a piece of data.

In our case, shared lookups were OK because the likelihood of data changing while scripts are running is exponentially lower than data changing between a staging and a run.



Ideal when multiple machines are going in parallel

Still good for single machines, though, if you design as a sort of mailbox system. Script A can leave a message for script B.



We attempted to use clustering to solve our problems with the integrity of our data.

Shared file storage

-lockfiles

-datafiles

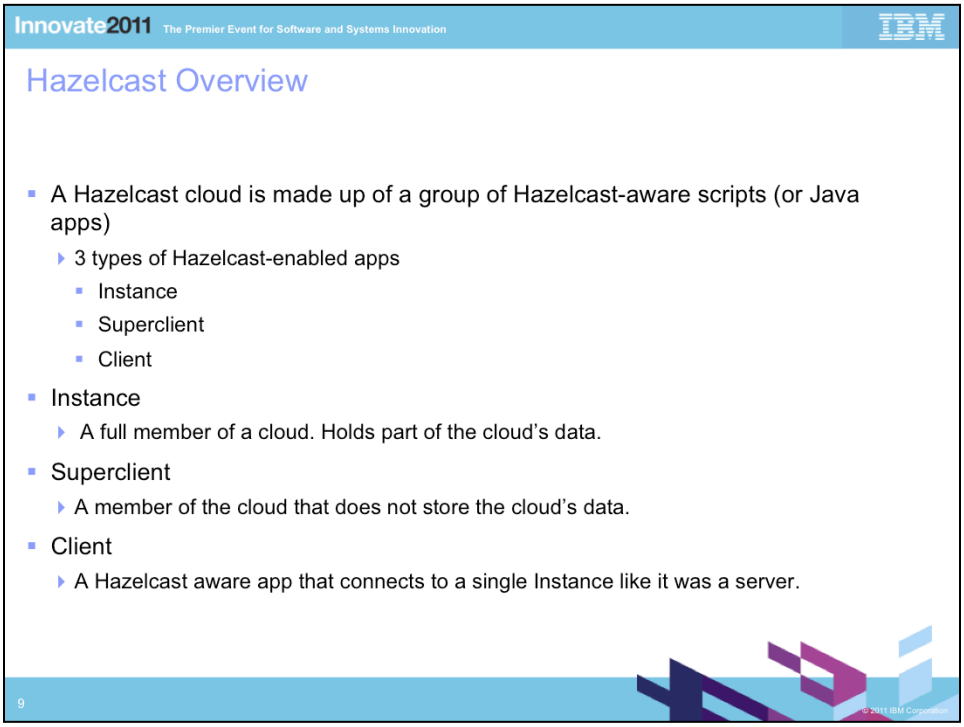
Database

temp tables

permanent tables

Hazelcast Overview

- www.hazelcast.com / hazelcast.googlecode.com
- Open source (Apache 2.0 License)
- Pure Java decentralized solution…just add the JAR to your classpath. No server needed.
- Similar to products like Terracotta, ehcache, and Gridgain but we thought it seemed easier to use
- All data is duplicated on at least two machines for redundancy
- RFT scripts on the same LAN can automatically find each other as long as they know the Hazelcast group name and password



Our configuration is relatively simple.

On each lab machine we have a standalone Java application that runs on machine startup as a Hazelcast "instance".

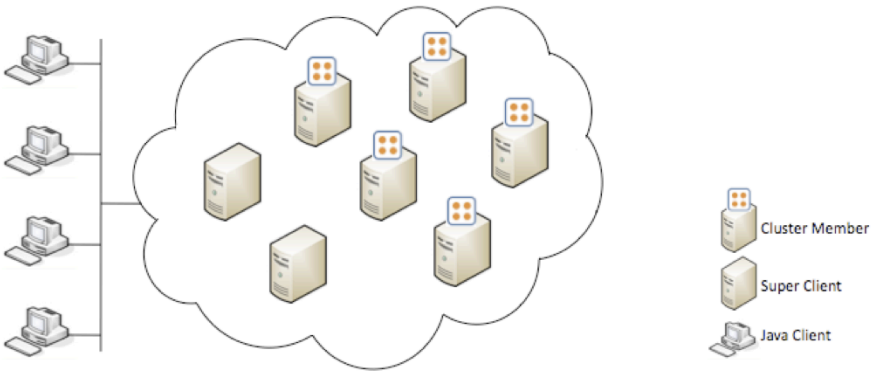
RFT scripts start up as "superclients" and connect to those instances.

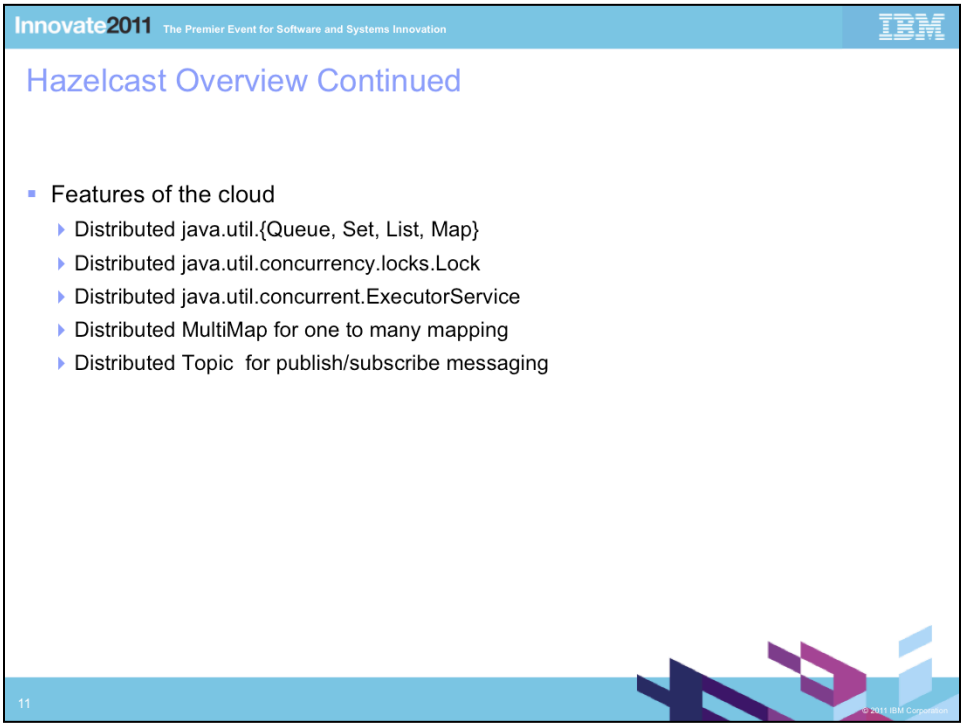
We made this design because there are stretches of time where our lab is completely devoid of running scripts and/or just stuff running on one machine.

A busy enough lab might be able to make RFT scripts run as "instances" and forego the standalone application.

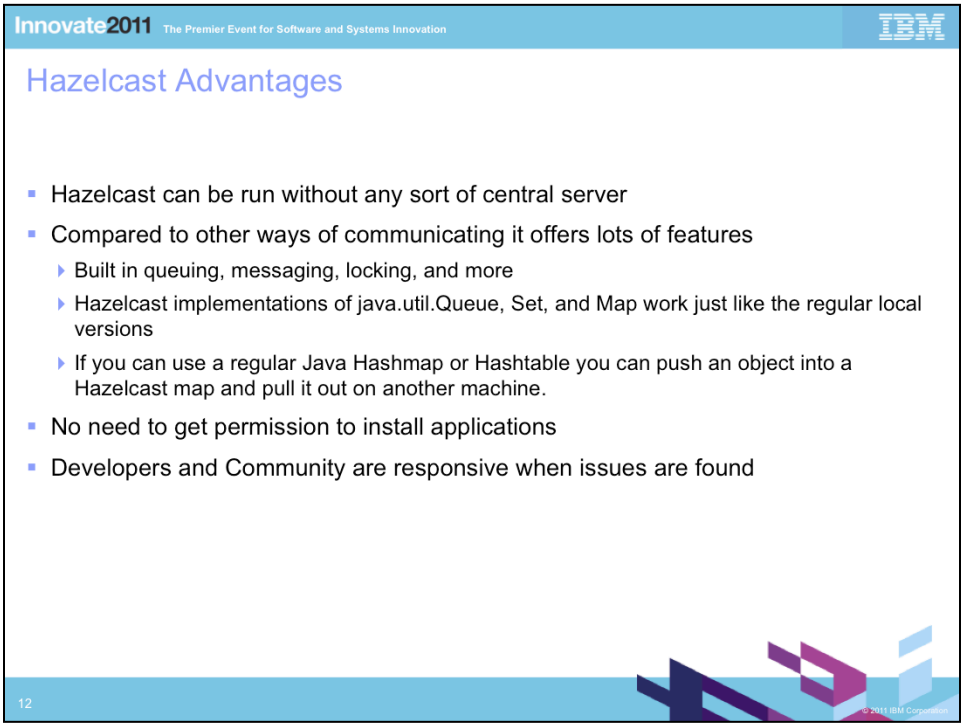
We chose to make our scripts "superclients" rather than "clients" because it made them extremely resilient to rebooting some of the instances, since they're connected to every instance at once.

Typical Hazelcast Cluster Design





See demo code for examples. We don't use Hazelcast's ExecutorService yet but we are looking into it.



Hazelcast development is moving along rather quickly. Between when I started the PowerPoint slides for this talk and when I finished them the dev team added a full distributed java.util.List implementation and fixed several issues we've encountered.

Hazelcast Advantages Continued

- The API is very easy to use.
  - If you have a Java class all you have to do is add “implements Serializable” and it’s usable with Hazelcast
    - E.g. “*public class Foo {*“ just has to become “*public class Foo implements Serializable {*“
      - Now Foo is ready to be put into Hazelcast structures and passed around your RFT machines

Hazelcast Disadvantages

- MultiMap and other new features have stability issues
- Network instability and/or overzealous antivirus software can drive a cluster nuts
- When the last member of the cloud signs off the cloud has effectively vanished (and so has the data)
  - We made a little Java app that we leave always running to ensure that the cloud is always there even if no RFT scripts are running.
  - There is an API you can code to if you want to provide a way for Hazelcast to persist data to a database.
- Limited Access Control
  - If certain data needs to be protected more than others (e.g. passwords) Hazelcast is not right for it.
  - A node that knows the group password can see everything on the group
- Harder to load data into from 3rd party sources (e.g. manual testers or apps not written in Java)

Database as a script communication tool

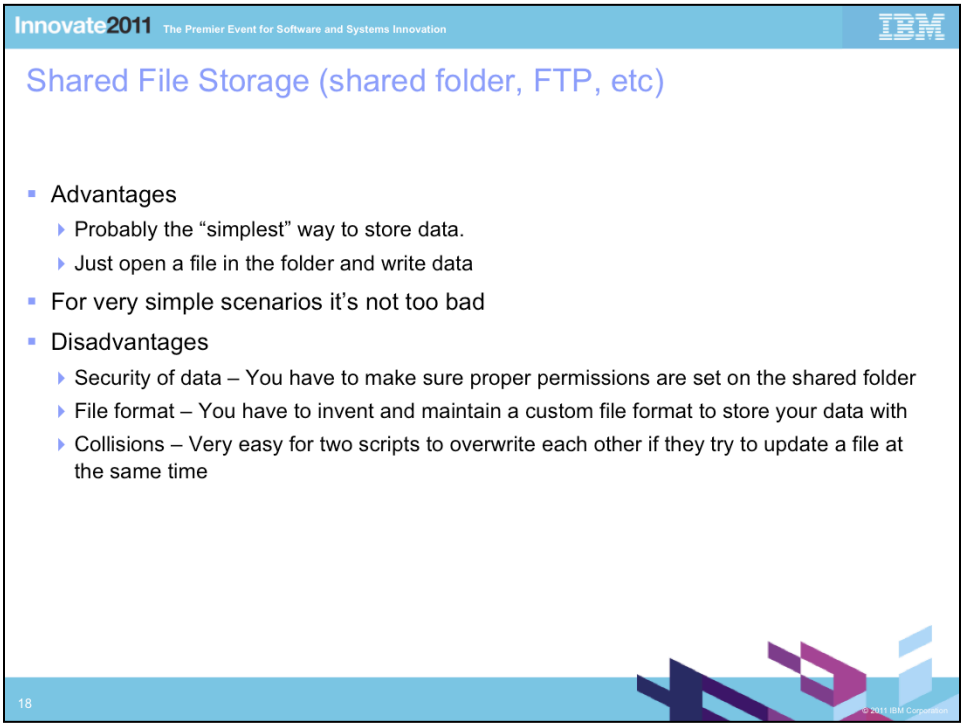
- We actually use a MySQL database for certain needs
  - Currently:
    - Passwords
    - Logs
    - Data that manual testers frequently update
- Any SQL database could work, but MySQL is free

Database Advantages

- Fine-Grained Access Control
  - Easy to restrict access to certain things (e.g. passwords) to specific users
- Easier for non-Java applications to update than Hazelcast
- Persistent
  - Data doesn't go away unless you actively erase it

Database Disadvantages

- Need to know SQL
- Hard to dump arbitrary Java classes into a database
- Might need IT department permission to run a “server”
- Single Point Of Failure
  - Or expensive and time consuming to get clustered database servers going



For our earliest queue counter tests we used a shared file that we literally wrote just the queue count into. We quickly outgrew this as we added testcases…the file was being updated way too often.

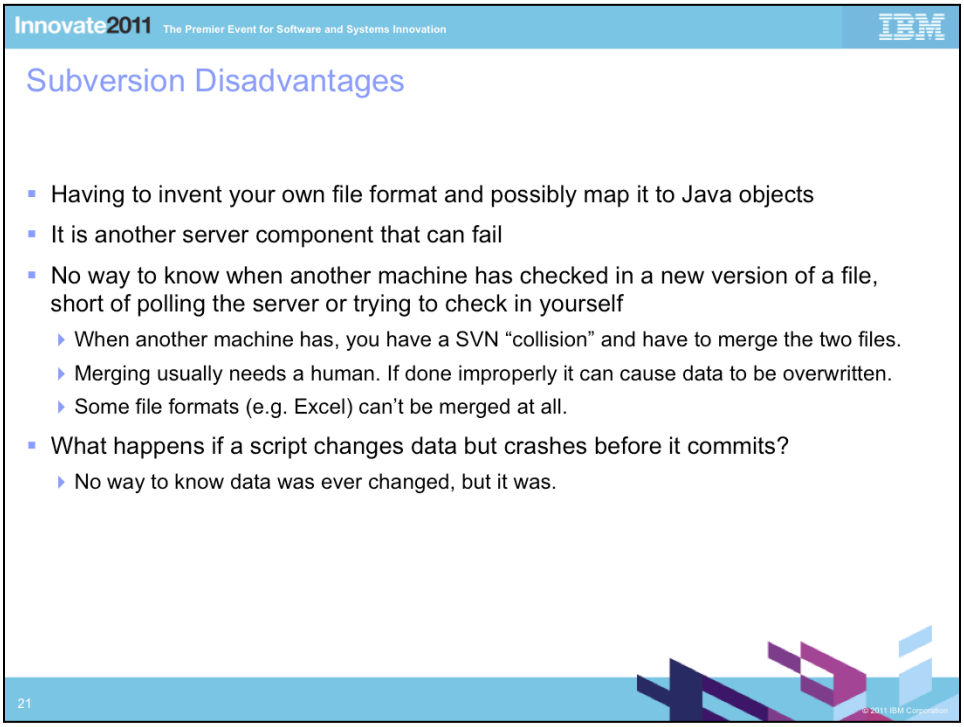
However, if you have something like 3 total machines and they update the shared file no more often than every couple minutes a shared folder might be all you need.

Subversion

- Subversion (SVN) is a commonly used document version control system.
  - Mostly used for source code control for open source projects
  - Similar to Clearcase, TFS, etc
- You can store any sort of file you want in it.

Subversion Advantages

- Using it as a store for data files has the unique advantage of providing a historical record of data
  - Each change is a SVN revision
- No worrying about scripts overwriting each other's data, unlike shared folders



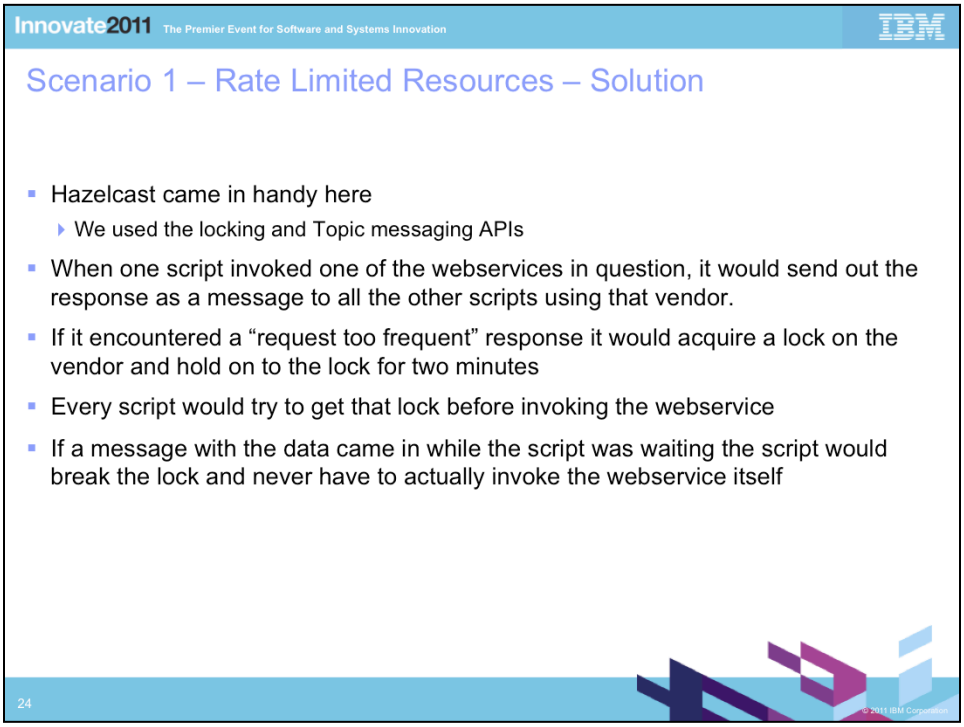
Excel files can be merged, if “merging” means opening both side by side and copying and pasting data from one to the other…but that requires a human.

Email Mailbox

- We've seen this done.
- It's like a shared folder, but done over a different protocol.
- It may seem odd but a previous coworker used this as a data sharing system.
  - He claimed it was more reliable for writing data than a shared folder
- Advantages
  - Very easy to see what data is being shared….simply open your email client of choice.
  - Reliable timestamps on every message
- Disadvantages
  - Mailbox can fill up, even due to external sources (e.g. spam)
  - Extra server that could fail
  - IMAP / Outlook MAPI protocols require additional libraries and coding time compared to shared folders or Hazelcast.

Scenario 1 – Rate Limited Resources

- LPS Desktop DIS batch download web services
  - All Desktop activity involving a vendor (aka attorney) gets queued up as XML for download with this service
  - This service brings back up to a certain number of items each time
  - Server limits you to one download every few minutes
  - Multiple machines would generate traffic and try to hit this service to make sure their data got added to the queue
  - Very easy for scripts to falsely fail because machine A happened to download machine B's data in addition to its own



This may not sound that simple, but parallel programming never is, regardless of what's running in parallel.

When scripts started they would register a MessageListener object with the queue named (webservice) + (company name)

Whenever some other script using the same company tried to invoke the same webservice the other script would send out the response as a message to this queue.

All scripts working with that vendor and web service could then check the response to see if it contained their test data.

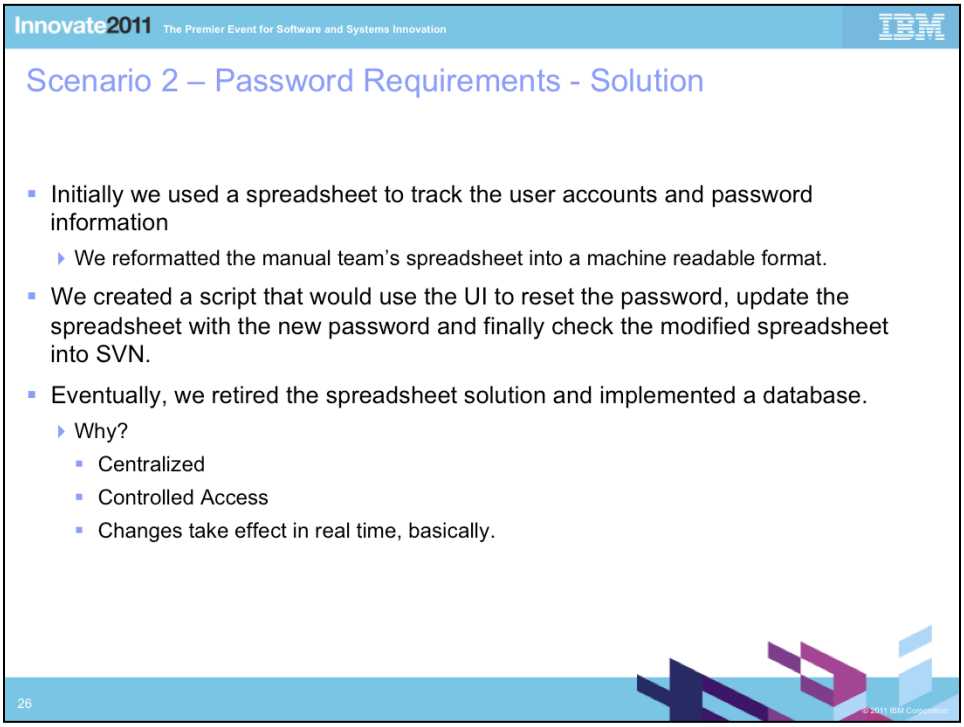
If it did they wouldn't have to invoke the web service themselves.

Why not a database?

1. Hassle of implementing a locking system to limit a vendor's downloading to once every two minutes
2. Hassle of converting Java objects to database tables.
    A. Already converted XML web service response to Java object.
3. No way to notify every machine that a vendor had hit this method.

Scenario 2 – Password Requirements

- In all LPS Desktop applications, security rules require passwords to change every 90 days (or less)
- Passwords from production testing accounts are supposed to be closely guarded by the persons / teams they are issued to
- We have hundreds of accounts on each environment for each Desktop application.
  - Thousands of accounts total. Too large for manual password management.



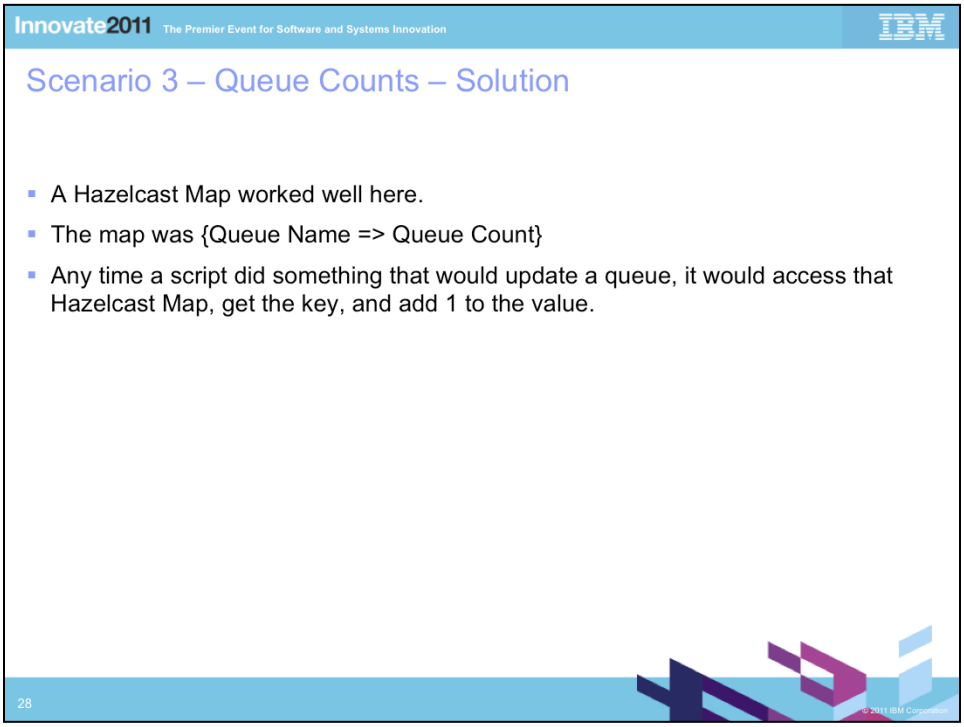
- Why not use SVN?
  - There is window of time where the modified spreadsheet is the only source with the true password.
  - What if the script fails mid-run before it was able to commit?
  - If we archive our old SVN code we're associating it with stale passwords.
    - Therefore if we have to run against an old version of the application we have to update our passwords manually and/or refresh the file from the current codebase.

- Why not use Hazelcast?
  - Security. No way to finely control access to the passwords….these are passwords we're talking about here.
  - Persistence isn't 100% guaranteed. Hazelcast doesn't save data to disk without a lot of coding.

- Why not just update the passwords in the app's database?
  - This would require database write access or coordination with DBAs.
  - Wouldn't work if the app was using Active Directory or similar as its password

Scenario 3 – Queue Counts

- LPS Desktop PM is a workflow management application
- Most of the loans in it go through work queues
- When scripts running in parallel on multiple machines do actions that would add or remove items from the queues they can notify each other (or at a minimum update a shared count of items in queue)
- Some of our test scenarios involve verifying that the queue count is correct, so we need to monitor all activity



No matter how you share the current count, you have to take the time to study the business rules to figure out what causes the queue count to change.

Why not use a database?

This data is too transient to be worth putting in a database or hassling with JDBC.

You would be spending too much time executing UPDATE statements to update this value.

Why not use a shared file?

Too easy for one machine to overwrite another's data. No locking at all.

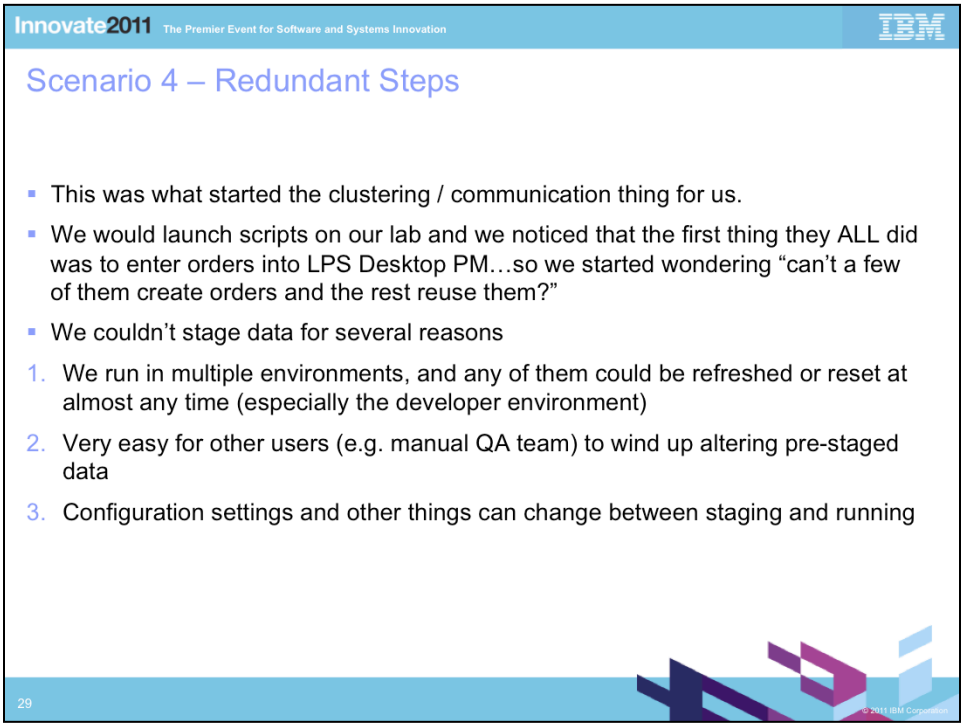
Why not use SVN?

No need to track this changing over time…also no way for machines to know they need to update.

Why use Hazelcast?

Hazelcast is fastest at updates…and it also automatically runs them in another thread…and it offers nice locking APIs if you're truly paranoid.

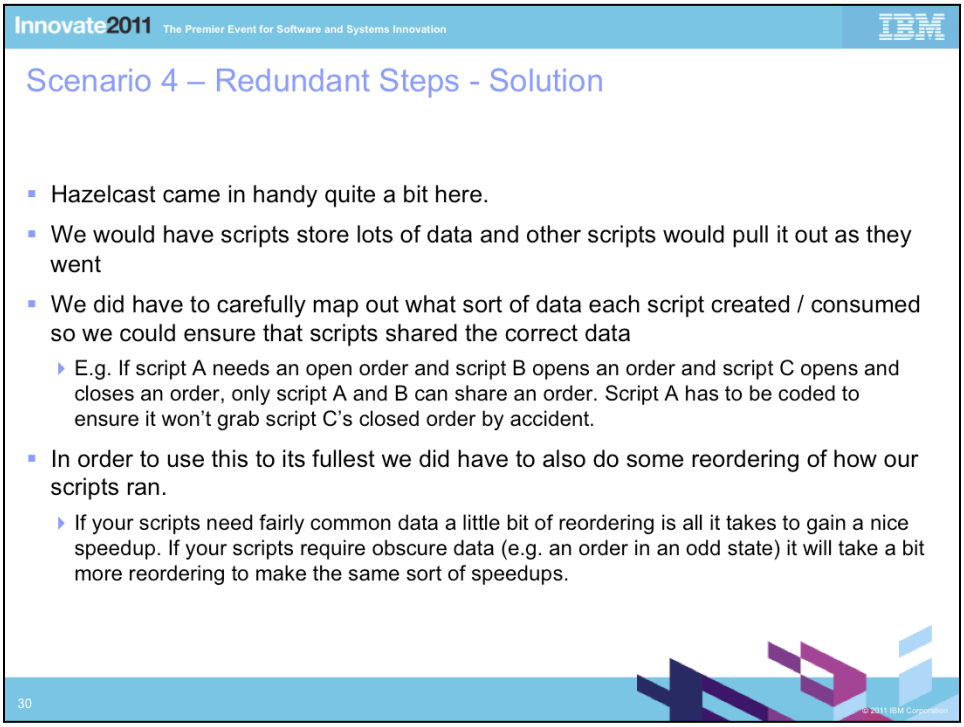
Also, Map.put() is much easier to deal with in terms of development effort versus



When it comes to ensuring configuration settings, sometimes automating the configuration steps is an acceptable workaround.

If you're lucky enough to have write access to your pre-prod environment databases a few SQL UPDATE statements can ensure your configurations are correct in a matter of seconds. This is also massively more stable than scripting the configuration UI.

Unfortunately we do not have write access to any of our apps' databases.



Why not a database?

1. The data doesn't need to be permanent.
2. The data doesn't need to be secured.
3. The nature of the data is too fluid to easily fit into a database table (or collection of tables)

A. If not careful you could mistakenly recreate the application's database.

Why not shared files?

1. Massive development effort required to create the file formats necessary to store all the information we might deal with.
2. No way to lock particular items.
3. No easy way to store metadata about our data.

Why not SVN?

1. No need to track the history of the test data
2. (See shared file reasons)

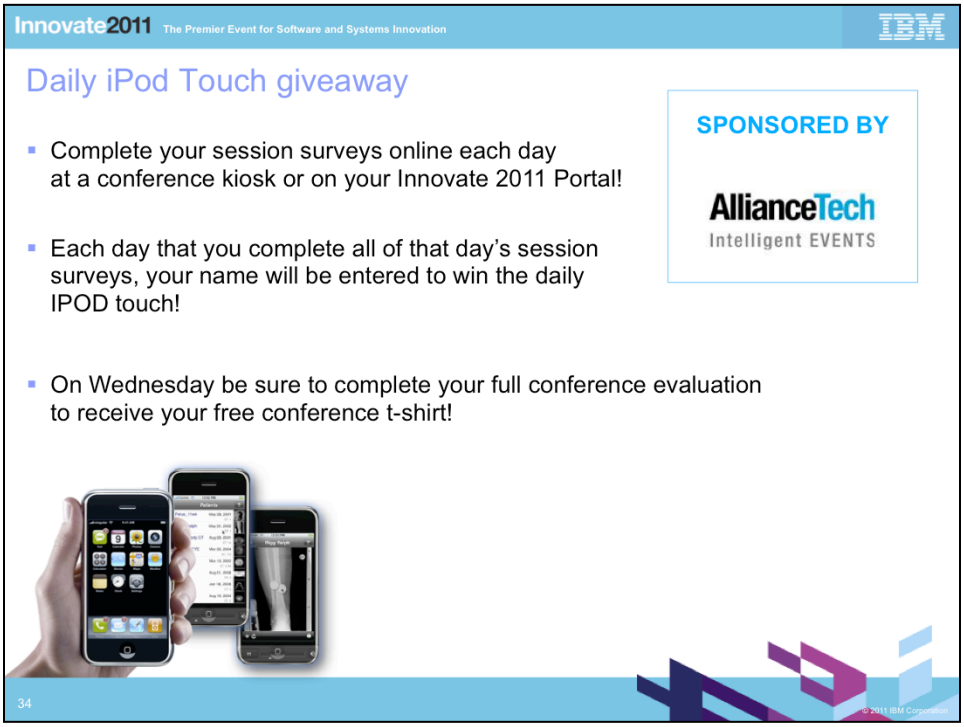
Why Hazelcast?

Author Note: Optional Rational QUESTIONS slide. Available in English only.

www.ibm.com/software/rational

Author Note: Optional Rational DEMO slide. Available in English only.

Author Note: Optional Rational QUESTIONS slide. Available in English only.



Daily iPod Touch giveaway sponsored by Alliance Tech

- Each time you complete a session survey, your name will be entered to win the daily IPOD touch!
- Complete your session surveys online each day at a conference kiosk or on your Innovate 2010 Portal!
- On Wednesday be sure to complete your full conference evaluation to receive your free conference t-shirt!

Author Note: **Mandatory Rational Closing Slide** (includes standard legal disclaimer). Available in English only.