SYBASE®

SCHEMA-DRIVEN EXPERIMENT MANAGEMENT

DECLARATIVE TESTING WITH “DEXTERITY”

DAN FARRAR
SQL ANYWHERE ENGINEERING

JUNE 7, 2010

MOTIVATION

WHAT'S WRONG WITH TEST SCRIPTS?

SQL ANYWHERE

- An RDBMS designed with the goals of:
  - Performance out of the box
  - Zero-to-low administration
  - Flexible deployment (OS, architecture, platform)

- As a result, SQL Anywhere contains many adaptive algorithms and self-adjusting parameters

- High performance code may contain timing-sensitive faults that must be found before it is shipped

TESTING THE SQL ANYWHERE ENGINE

A complex RDBMS requires many levels of testing.

- Unit testing
- Functional testing
- System testing

- Regression tests define well-known test cases.
- Need to hunt for previously unknown classes of bugs, too.
- Repros for timing-sensitive bugs are very hard to obtain

- Tuning a database configuration requires a similar mix of guesswork and analysis
- The first 80% of performance is easy; the last 20% is not

A SAMPLE SCRIPT

Even a simple batch file has dozens of variables it could range over.

```
dbspawn.exe dbsrv11.exe -gn 25 -c 200m
 asatest.db
fetchtst.exe -t SCROLL -js 10 myqueries.sql
 > res.out
dbstop.exe
```

What if a user wanted to see what the effect of:

- -changing the server or dbtest command line?
- -setting environment variables that control server behaviour?
- -setting database options?
- -using a specialized build of the database or the server?

AD-HOC TESTING

The search for new bugs reproducibles and new optimal configurations.

- Configuration explosion
  - Covering the search space complete is not feasible
  - Testing tools must help human testers plan and cover the search space intelligently…
  - …and **remember** what was covered and how

- Many experimental tasks require human experimenters to wait to:
  - Build custom engines
  - Copy large files
  - Shut down and reconfigure the test
  - Recover from errors (i.e. file already exists)

DESIGN GOALS AND PRINCIPLES

RDBMS + MAKE = DEXTERITY

DESIGN GOALS

Cluster the operations where human attention or decisions are needed.

- Allow a single user to manage ad-hoc experiments on many machines simultaneously
- Concentrate human time to the test phases where human attention is needed:
  - Test specification
  - Analysis of results
- …and let machines do the rest.
- Make sure that no experimental results are wasted; document everything, so that it can be used in later analysis.

- **Main Goal: Combine the persistence and data manipulation of an RDMS with the fire-and-forget simplicity of the MAKE utility**

DESIGN PRINCIPLES

A declarative design provides good composability of experimental steps.

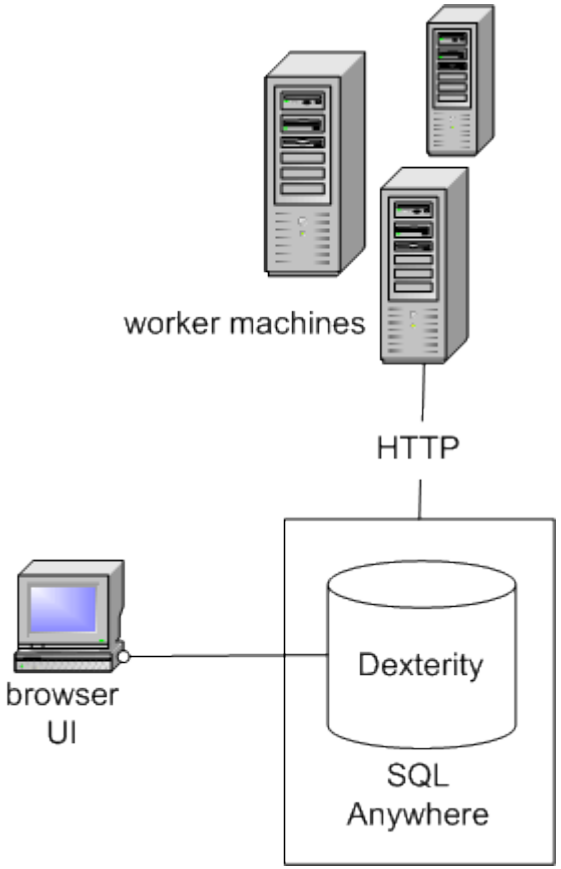
- Everything is managed by the server
  - Test scheduling and analysis is easiest this way
- Everything is declarative
  - Because a user always specifies what results she ultimately wants, deliverables can be composed in any desired fashion
- Everything is append-only
  - Even failed runs are available for inspection
  - There is no doubt about what state a resource is in

ARCHITECTURE

ARCHITECTURE

As with all automation, allow the human to work smarter, not harder.

- SQL Anywhere includes an integrated web server
- The “Dexterity” system itself lives within a SQL Anywhere server
- All configurations and deliverable objects are stored in the Dexterity database
- Worker (client) machines can ask for work and run it when all its prerequisites are available
- Dependencies are modelled as foreign keys in the database



SYBASE®

CONFIGURATIONS

All information that might ever vary should be stored in the database.

- Configuration rows represent the independent input parameters to the testing process

- Configurations generally never change; if they need to, a new configuration must be created

- Example configurations:
  - Build type
  - Build architecture
  - Machine name, memory, CPUs

RESOURCES

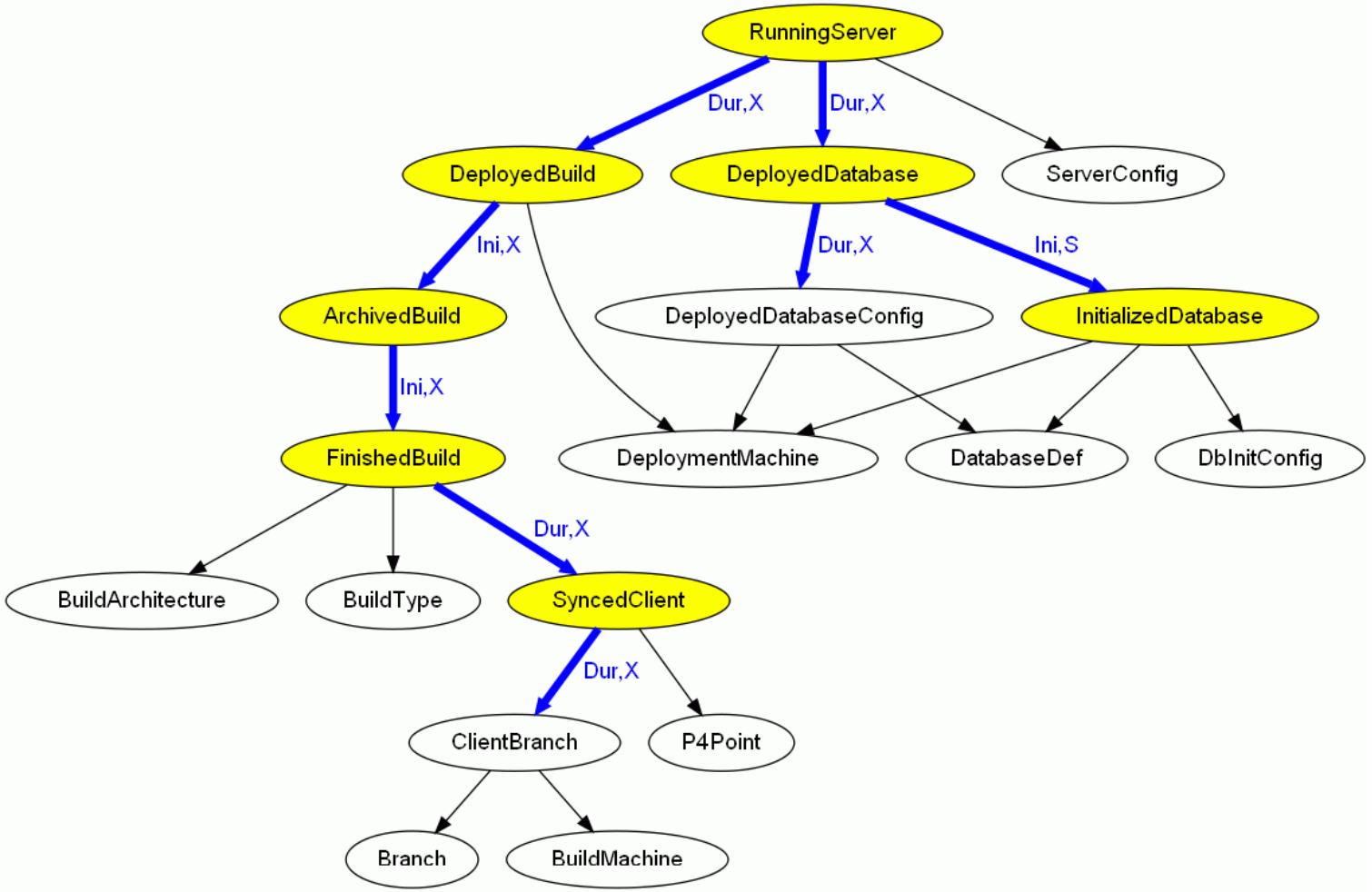
All relevant output of test processes must also be stored in the database.

- Resource rows represent the intermediate and final deliverables of an experiment.

- Two types of resources:
- Init-Only: Resources that require work to instantiate, but do not require a teardown step
  - Ex., SyncedClient, FinishedBuild
- Init-and-Fini: Resources that require an action to create/instantiate, and to tear down
  - Ex., RunningServer

CORE DEPENDENCY GRAPH

A deliverable is primarily defined by all its dependencies.



SYBASE®

SCHEDULER

The scheduler does what needs to be done, but no more than that.

- Server is aware of state of all resources
- It is also aware of the dependencies between these resources, and how long those dependencies last

- At client request, server can query dependency graph, looking for
  - resources that are ready to initialize,
  - resources that must be finished to allow a new resource to be instantiated

TEST CLIENTS

Test clients should know how to do their immediate job and no more.

- Clients are Python scripts that communicate with stateless, RESTful web-services
- A test process can be halted (intentionally or not) and resumed without losing work
- Data is moved in JSON format and is automatically marshalled

USER INTERFACE

The UI must permit the easy specification of test plans.

- The UI is web-based, for ease of deployment
- This also allows it to be tightly integrated with the DBMS

- The UI allows a user to enter resource rows describing what she wants instantiated
  - This remains just a request until the scheduler determines that all dependencies for the resource have been met
  - It can then be instantiated

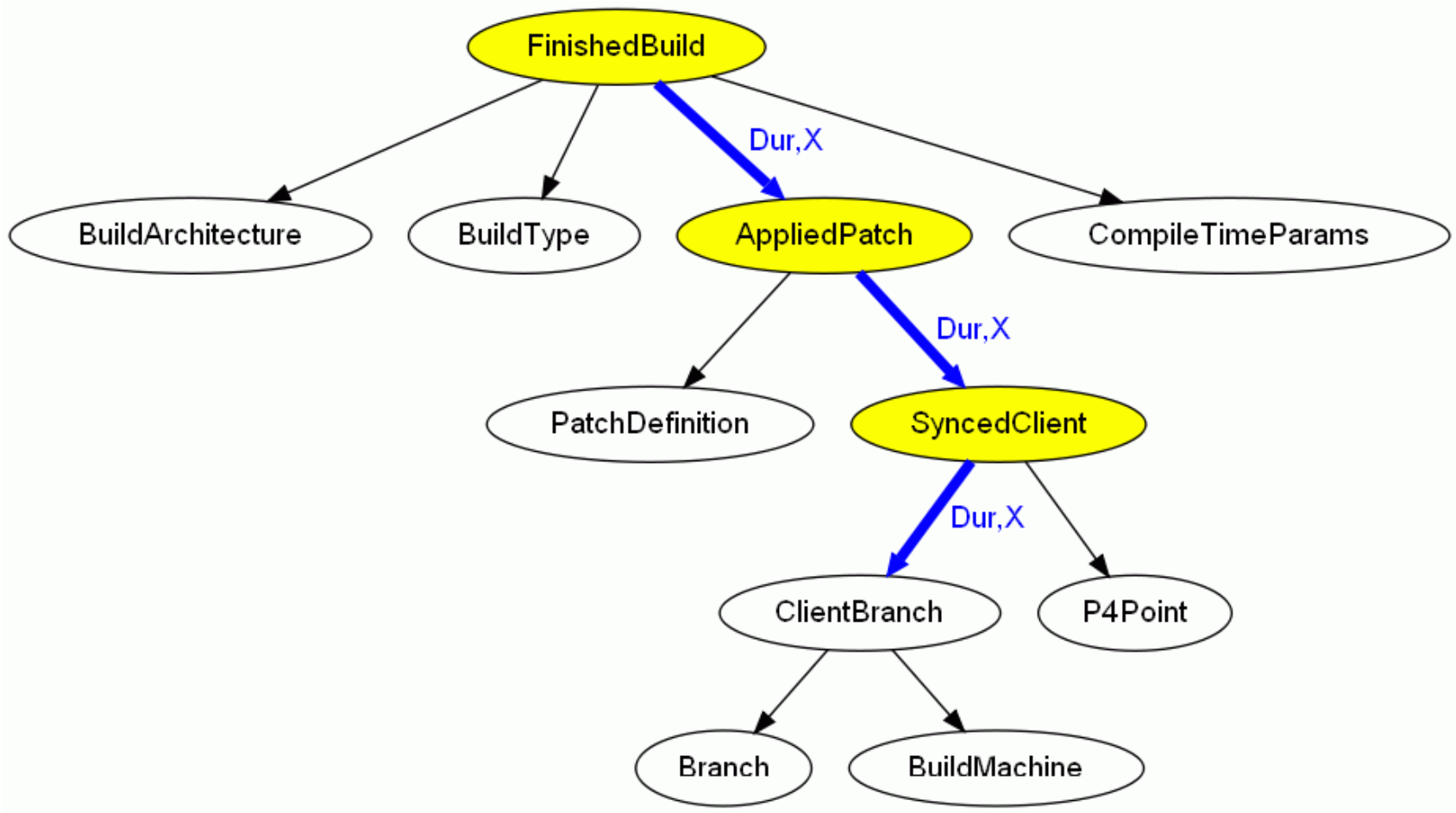
- The UI combines the operations of selecting, filtering, and inserting data

DEMO

TEST DEFINITION AND ANALYSIS

EXTENSIBILITY

New independent variables are easy to add without invalidating previous results.



SYBASE®

FUTURE WORK

- Provide a prettier interface
- Improved support for higher-level data manipulation
- More use time