Reinventing Data Import

DATABOLT

07/17/2023

About Me

James Dornan

Professional Background

- Changed careers from banking to computers at 21
- Early Innovations
  - Automation assisted data entry
  - Unattended database health alert system
  - National print & document archive system for Cigna
  - Business/Personal ISDN & modem ISP and hosting service

- Major Projects
  - Developed a web application and database to manage and display the entire company's organizational hierarchy, tracking employees, locations, and critical details from hire to termination.
  - Created an eBay-style auction portal for a Fortune 500 company, enabling branch managers to auction rental equipment, tracking profits and generating an estimated $100 million annually

Universities Are Just Like Businesses

Access control systems are not geared specifically for large schools. Maybe large campuses, but not schools.

Schools handle students far different than businesses handle employees, and students absolutely expect everything to move at TikTok speeds.

Schools are like businesses, except…

- They hire almost everyone in September and terminate them in June.
- Most employees' contracts gets extended each semester.
- A large population of employees live at work.
- Employees have unrealistic expectations.
- Those same employees are also the customers and pays to work there.

Rental Car Standard

Updates to OnGuard records should be as fast as renting a car/time machine.

How We Get Updates at Stanford

1. Connect to central data warehouse using Weblogic JMS client.
2. Receive JMS messages containing RefIDs that have been updated.
3. Download XML documents using RefIds one at a time.
4. Parse and process each XML document.
5. Modify data as required to conform with the card systems.
6. Send to receiving systems, like OnGuard.

Stanford University

Harvester System

Using a simple import program that just passes data into OnGuard.

- Data is updated even if no data has changed.
- Resource intensive, memory, CPU, network.
- Potential trigger activation when no action is actually required.
- Audit logs might not reflect actual updates.
- Excessive writes to storage media.
- Sequential record processing.
- 3-20 seconds per record, with average around 4.7 seconds.
- Slower when the system is under heavy load.
- Occasional failure of OnGuard or WMI.
- 10,000 records takes 13-17 hours to process.

Stanford University

Improved Harvester

Our improved system was much better.

- Hashing incoming data removed duplicate updates.
- Incoming data compared to existing records before updating.
- Only fields that change are updated.
- Sequential record processing.
- Slow. 2-5 seconds per record, with average around 2.5 seconds.
- Custom processing.
- Slower when system under heavy load.
- 10,000 records takes 7-10 hours.

Faster Meant Fewer Features

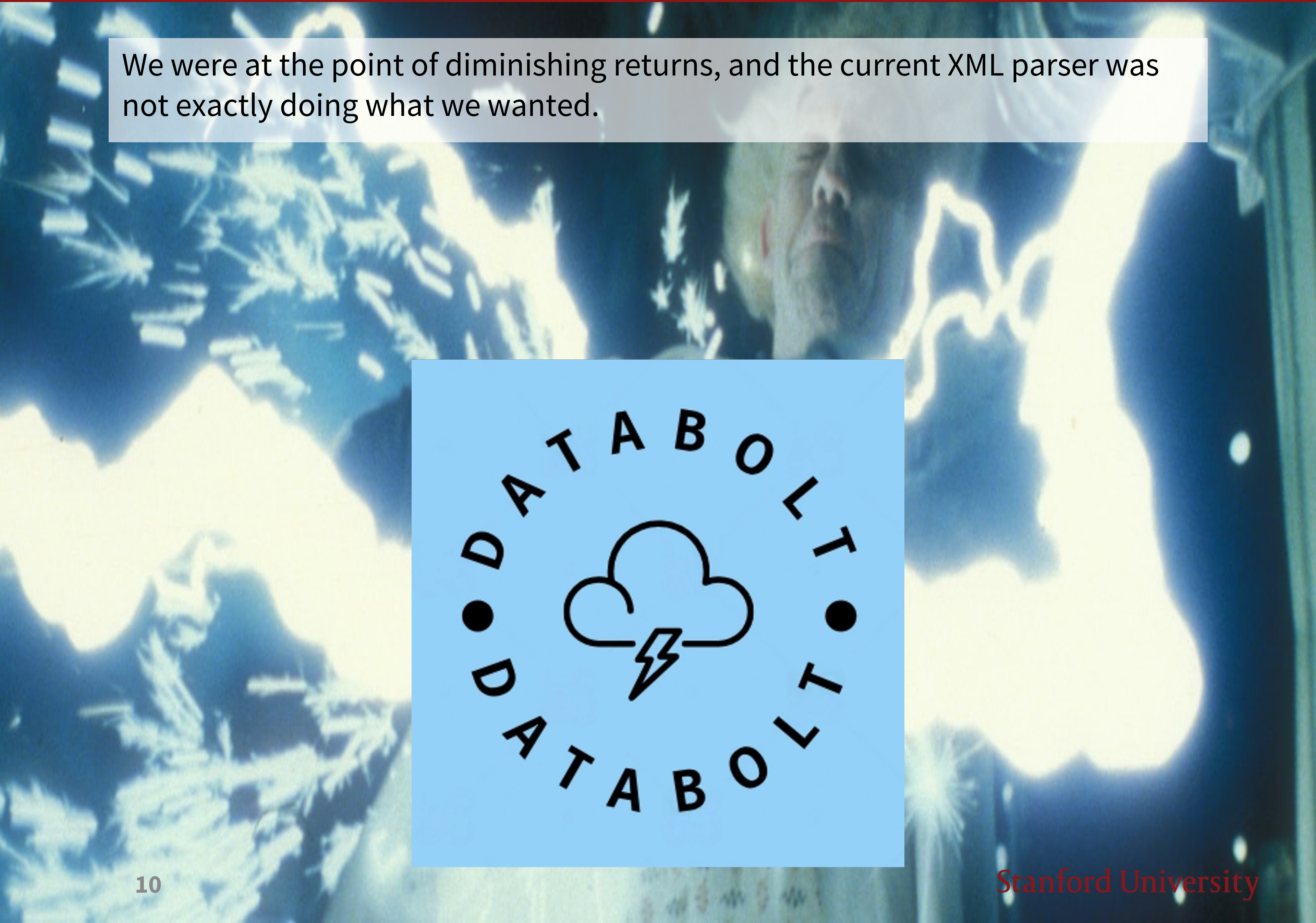
When adding features to processing incoming records the processing slowed down.

A modest increase in processing time, but anything multiplied by 10,000 was a long delay in our nano-second universe.

Speed And Features

Ultimately we'd taken things as far as they could go. We needed the ability add better processing of the XML records, add more features, and satisfy our customers. However, we'd almost need a time machine to do that.

A Different Approach



We were at the point of diminishing returns, and the current XML parser was not exactly doing what we wanted.

Design

Databolt is an advanced, containerized system designed for the efficient download, import, and processing of incoming and updating records for staff, students, and other person records. This Docker-based service comprises multiple containers, each handling specific tasks to ensure seamless integration with OnGuard, CS Gold, and Atrium.

- Containerized for portability and isolation.
- Multiprocess to leverage different languages and resources.
- Docker Compose Service to create a single service.
- Multithreaded for parallel processing.

In The Past

We processed incoming records one at a time, each record waiting until the last completed before being processed.

The Future Is Now

We will process records in parallel (threading). Since there is time spent waiting on outside resources, database, OnGuard, and HTTP the burden to the server is exceedingly low. Rather than waiting in line we're just going to process as many records at once as we can without breaking systems.

Stanford University

Docker Compose Service

With Docker Compose a series of Docker containers can be used as a single service. They can all be started or stop at once, act as a single unit, and communicate freely with each other while completely firewalled off from the rest of the world.

Databolt Containers

- **Receiver**, a Java based program running every 3 minutes that downloads a list of RefIds form our data warehouse, MAIS, and passes them to the Ingress container.
- **Ingress**, a Golang microservice that acts as a simple gateway to get information into and out of the Databolt set of containers. Incoming requests (commands) are passed to the Bolt listener thread.
- **Bolt**, a PHP 8.3 service that hosts a framework for splitting complex programs into multiple threads to allow for parallel processing.
- **Redis**, this is an in-memory database that is also written to disk. It is as fast as computer memory while also backed up to disk.
- **Gold**, this container reads record ment for CS Gold and writes them out to the CS Gold import folder where they are picked up.
- **Atrium**, same as CS Gold, but with a different file configuration, folder, and size limitation.

The totals size of these containers are about 224MB on disk and the Databolt service can use as much as 900BG of RAM under a heavy load. This is very light on resources.

Receiver

The Receiver is a simple Java JMS program that downloads messages that container the RefIds of records that have been updated.

It replaces nearly 30,000 lines of code that had change maintainers over the years and had become hard to understand with less than 500 lines of ease to maintain Java code.

- Runs every 3 minutes.
- Connects to MAIS.
- Downloads Refids.
- Checks for duplicates.
- Uses HTTP to pass them to the Ingress container.

Ingress

The Ingress container is a simple web server that takes requests in the form of a URL and is only accessible from the local machine. Each URL issues a different command to the Bolt service container. The only command supported at this time is “maisimport”. The URL is http://ingress/import/person/<RefId>

- Listens for HTTP requests.
- Passes the command to the Bolt container.
- Uses a simple in-memory message queue.
- If a reply is requested, it waits for the reply.

Redis

Redis is used to create a high-performance repository for data that can be shared between all processes. The data is stored on disk in the event that the machine should crash or the service should failure. In the event of a failure it will allow processing to resume from where it left off.

- Stores persistent data.
- High speed.
- Widely supported.
- Stable.

Stanford University

CS Gold & Atrium

Both of these containers are identical in their function. They read data intended for their respective system, generated by the Bolt container, from Redis and output into a designated import folder shared from the destination system.

- Runs every 5 minutes.
- Reads data from a Redis queue.
- Outputs to a file in a specific folder.

Bolt

The Bolt container runs a PHP 8.3 service that listens for incoming “command” requests and then places them into a Redis queue. A command is divided into tasks based on events. The default event is “Start”. Each task is run in a different thread.

- Configurable number of threads (default 100).
- Saves state at each stage of processing to resume after stopping.
- Highly optimized code.

Command Class

Command classes are named after the command that is sent to Databolt. At the moment the only command class is ‘maisimport’. Command classes controls which tasks are run in response to incoming events. They control the workflow for any given command.

If an event triggers a new task the data output from the last task is passed to the new one, allowing each task to contribute to the data accessible to downstream tasks, global data for all related tasks.

Tasks

Tasks are executed in

- Extensible workflow.
- Modular design.
- Up to 100 total task may run at once.
- Focused on doing only one thing.
- Reuses threads speeding up start time.
- Generates events when done and passes data to next task.

Stanford University

Features Of Doing 100 Things At Once

- **High Throughput:** Capable of processing up to 500-1,000 records per minute, Databolt ensures timely updates and data synchronization.
- **Resource Efficient:** Designed to run on minimal resources (1 CPU, 4MB RAM, 204MB storage), making it cost-effective and easy to deploy.
- **Event-Driven Architecture:** Automatically triggers various task processes as needed, enhancing responsiveness and efficiency.
- **Programmable Workflow:** Utilizes flexible “Command” classes to tailor workflows to specific needs and scenarios.
- **Parallel Processing:** Supports up to 100 threads, enabling concurrent task execution and reducing processing time.
- **Stateful Recovery:** Ensures continuity by resuming operations immediately after a system failure, minimizing downtime.
- **Containerized Service:** Built as a Docker service with multiple containers, Databolt provides isolation, scalability, and ease of deployment.

Stanford University

Benefits Of Doing 100 Things At Once

- **Scalability:** Easily handles large volumes of data, perfect for institutions with growing records.
- **Accuracy:** Reduces errors through automated processes and real-time updates.
- **Cost Savings:** Operates on low resources, reducing infrastructure costs.
- **Reliability:** Robust architecture ensures data integrity and availability even in case of failures.
- **Ease of Deployment:** Docker-based architecture simplifies deployment and management, allowing quick setup and consistent environments.
- **Extensibility:** Easily extended to incorporate additional functionalities or integrate with other systems as needs evolve. We can add almost anything Krystal wants.

Stanford University

Review

Using a multi-threaded approach we are able to download, process, and update records at a rate of 8.33-16.66 records per seconds rather than 2.5-7 seconds per record under the old system. This is an increase of 21 times the previous speed.

10,000 records will now download, process, and update in 10-20 minutes. 500-1,000 per minute.

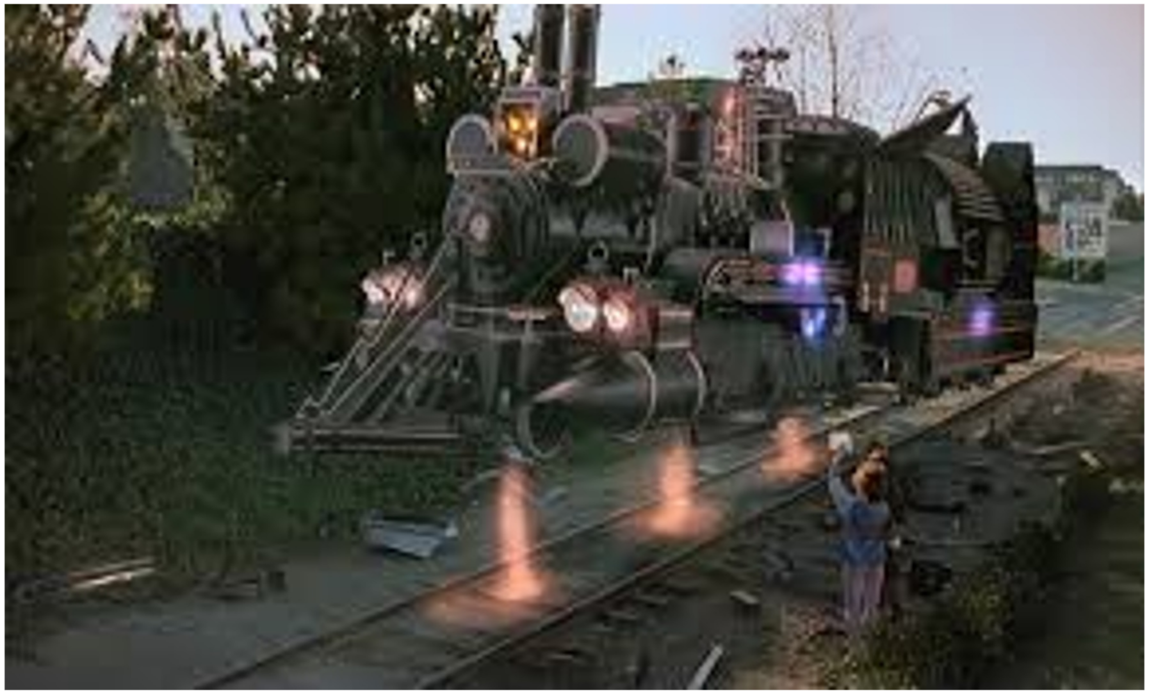
With the processing of records no longer directly impacting the processing of other records we can add more complex or specialized transformations to the data.

The modularity of the design allows interfacing with other systems without any impact to existing code.

Stanford University

Future

- Move to OpenAccess
- Write logs to Redis in memory and have a logging process write to disk.
- Create RactJS Web UI to monitor and control.
- Have Receiver write directly to Redis queue.
- Experiment with increasing overall Tasks to 150 at a time.
- Everything Jay wants. All hail Jay.
- Everything Krystal wants. All hail Krystal.



Geek Stuff - Docker Container Sizes

These are the sizes of the containers. I attempted to get them as small as reasonably possible.

Docker Image	**Size**
Atrium Import	101KB
Gold Import	101KB
Redis	827KB
Ingress	8.31MB
Receiver	76.4MB
Bolt	135MB
Total	**220.74MB**

Stanford University

Geek Stuff - Technology

- Docker, docker compose
- PHP 8.3
- PHP Parallel
- ZMQ
- Golang
- Java
- Weblogic
- Redis
- C++
- Alpine Linux