Go a Jump in a Lake

Andrew Fryer @deepfat
Big Data Week Conference 2016

Finance  Healthcare  Retail  IoT  Entertainment  Travel

NEW  NEW  NEW  NEW  NEW  NEW

@bdw16
Think about a really big data problem
Tough Love for Microsoft Search

Danny Sullivan, Search Engine Land
December 30, 2008

Google wasn’t just a brand, it was a habit

bing comes from behind
Processing hundreds of billions of documents

Understanding billions of entities

Reasoning trillions of entity relationships

Seeing hundreds of billions of queries and clicks

Serving and adapting for billions of users
Microsoft had a sliding US search share 2005-2009
Why
Hard to hire data scientists

Lots of people involved need access to data

Huge technical integration challenges

Massive volume, velocity, and variety of data
Laser focused on measuring relevance

Measure the rate of exploration

Empower everyone to explore (100x people)
Built an exabyte-scale data lake for everyone to put their data of all types

Built tools that could be used by any developer

Built machine learning tools for collaborating across large experiment models
Microsoft doubles search share

We went through these learnings

It is a hard problem

It requires a lot of investment and know-how to get it right

With Cortana Intelligence you don’t have to be the size of Bing to solve the problem

Where does this leave you?
Cortana Intelligence Suite

Information Management
- Data Factory
- Data Catalog
- Event Hubs

Big Data Stores
- Data Lake Store
- SQL Data Warehouse

Machine Learning and Analytics
- Machine Learning
- Data Lake Analytics
- HDInsight (Hadoop and Spark)
- Stream Analytics

Intelligence
- Cognitive Services
- Bot Framework

Dashboards & Visualizations
- Power BI

Data Sources
- Apps
- Sensors and devices

Data
Intelligence
Action

People
Apps
Web
Mobile
Bots
Automated Systems
Data at Scale in CIS

Information Management
- Data Factory

Big Data Stores
- Data Lake Store
- SQL Data Warehouse

Machine Learning and Analytics
- Data Lake Analytics
  OR
  HDInsight (Hadoop and Spark)

Intelligence
- Power BI

Dashboards & Visualizations

Data Sources

Data Sources

Microsoft
Azure Data Lake

Analytics
- HDInsight ("managed clusters")
- Azure Data Lake Analytics

Storage
- Azure Data Lake Storage
Azure Data Lake
Azure Data Lake
Store & managed clusters

On-premises

Map Reduce  Hive, Pig  HBase  Storm

YARN-based Compute
HDFS/WebHDFS API
Hadoop File System

Hadoop Cluster

Cloud

MapReduce  Hive, Pig  HBase  Storm

YARN-based Compute

Azure Data Lake managed clusters

WebHDFS API  Hadoop File System

Azure Data Lake store

Microsoft
Azure Data Lake

Analytics Service
Azure Data Lake Analytics Service

A new distributed analytics service

- Built on **Apache YARN**
- **Scales dynamically** with the turn of a dial
- **Pay by the query**
- Supports **Azure AD** for access control, roles, and integration with on-prem identity systems
- Built with **U-SQL** to unify the benefits of SQL with the power of C#
- Processes data **across Azure**
Developing big data apps

- Author, debug, & optimize big data apps in **Visual Studio**
- Multiple Languages **U-SQL, Hive, & Pig**
- Seamlessly integrate **.NET**
Analytics: Two form factors

**HDInsight**
- Managed Hadoop clusters
- n1, n2, n3, n4
- Hive/Pig/etc. job

**ADLA**
- Analytics service
- Lots of containers
- U-SQL/Hive/Pig job

**Storage**
- Blobs or ADLS

**Input**

**Output**

YARN Layer
ADLA Account
Azure Data Lake

U-SQL
What is U-SQL?

- A **hyper-scalable**, highly extensible language for preparing, transforming and analyzing all data
- Allows users to **focus on the what**—not the how—of business problems
- Built on **familiar languages** (SQL and C#) and supported by a fully integrated development environment
- Built for **data developers & scientists**
The Origins of U-SQL

Next generation large-scale data processing language combining

- The declarative, optimizable and parallelizability of SQL
- The extensibility, expressiveness and familiarity of C#

High performance  Scalable  Affordable  Easy to program  Secure
U-SQL Philosophy

Achieve the same programming experience in batch or interactive

- Schematizing unstructured data (Load-Extract-Transform-Store) for analysis
- Cook data for other users (LETS & Share)
  - As unstructured data
  - As structured data
- Large-scale custom processing with custom code
- Augment big data with high-value data from where it lives
U-SQL language philosophy

Declarative query and transformation language:
- Uses SQL's SELECT FROM WHERE with GROUP BY/aggregation, joins, SQL Analytics functions
- Optimizable, scalable

Operates on unstructured & structured data
- Schema on read over files
- Relational metadata objects (e.g. database, table)

Extensible from ground up:
- Type system is based on C#
- Expression language is C#

User-defined functions (U-SQL and C#)
User-defined types (U-SQL/C#) (future)
User-defined aggregators (C#)
User-defined operators (UDO) (C#)

U-SQL provides the parallelization and scale-out framework for user code
- EXTRACTOR, OUTPUTTER, PROCESSOR, REDUCER, COMBINERS

Expression-flow programming style:
- Easy to use functional lambda composition
- Composable, globally optimizable

Federated query across distributed data sources (soon)
Expression-flow programming style

- **Automatic** "in-lining" of SQLIP expressions – whole script leads to a single execution model
- Execution plan that is **optimized out-of-the-box** and w/o user intervention
- Per-job and **user-driven** parallelization
- Detail **visibility** into execution steps, for debugging
- **Heat map** functionality to identify performance bottlenecks
U-SQL overview
E-commerce scenario: Before

- Web server farm
- Transactional database servers
- Users
- Server logs
- Custom file format
- CSV
- Combined data
- Customer purchase records
- Custom aggregations
- Reports
- Custom file format
- Server logs
- CSV
- Combined data
- Customer purchase records
- Custom aggregations
- Reports
E-commerce scenario: Sample web log data

Encrypted user ID                  Start time                          End time                  Region                  Comma separated list of pages


<B4332&8*_ŞR>  <2/15/2012 11:53:16 AM>  <2/15/2012 11:58:32 AM>  <en-US>  <en.wikipedia.org/wiki/Weight_loss;webmd.com/diet; exercise.about.com>  <82,5,6,7,34,56,5678>

<XY54#@asd9>  <2/15/2012 11:54:16 AM>  <2/15/2012 11:56:17 AM>  <en-gb>  <microsoft.com; Wikipedia.org/wiki/Microsoft; xbox.com;msn.com;www.yahoo.com>  <87,92,45,53,45,45,48,98,24>


<MGD%Ş@78JH>  <2/15/2012 11:55:16 AM>  <2/15/2012 11:58:36 AM>  <en-us>  <skiresorts.com;ski-europe.com;www.travelersdigest.com/ski_resorts.htm>  <2,45,56,4,6,9,65,98,24>

<OPO*BSD%S>  <2/15/2012 11:56:16 AM>  <2/15/2012 11:59:45 AM>  <en-fr>  <running.about.com;ehow.com;go.com’nike.com;nfl.com>  <1,8,72,34,89,34,27,48,67>
<table>
<thead>
<tr>
<th>User ID</th>
<th>Time</th>
<th>Product IDs</th>
<th>Total amount ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>78229021</td>
<td>2/15/2012 11:53:16 AM</td>
<td>SR27821, CO98241, HG4214</td>
<td>214.50</td>
</tr>
<tr>
<td>999645224</td>
<td>2/15/2012 11:53:16 AM</td>
<td>SRT242421, VFG3243, TR3253, BET353, OPB236, FE4365, KL5634, HI4634, MI4634</td>
<td>4,213.78</td>
</tr>
<tr>
<td>397633634</td>
<td>2/15/2012 11:54:16 AM</td>
<td>A3256</td>
<td>58.67</td>
</tr>
<tr>
<td>8342834114</td>
<td>2/15/2012 11:54:16 AM</td>
<td>A8427</td>
<td>44.42</td>
</tr>
<tr>
<td>2421412</td>
<td>2/15/2012 11:56:16 AM</td>
<td>LW04682, MJ54655</td>
<td>305.75</td>
</tr>
</tbody>
</table>
E-commerce scenario: Challenges before

Needed a future-proof storage solution to hold many PBs

- Open Source big data analytics tools have a steep learning curve
- Even with scale out, reporting time increases as data volume increases
- Home-grown scale-out frameworks are difficult to develop and maintain
E-Commerce scenario: After

**Web server farm**

**Azure Data Lake**
- Server logs
- Custom file format

**Custom file format**

**Azure SQL DB**
- Customer purchase records

**Azure Data Lake Analytics**

**Reports**
- Custom aggregations
  - U-SQL analytic app

**Transaction database servers**

**Users**
Why would you use it?

Reduce learning curve

**SQL C#**

U-SQL is based on SQL and C#

Reuse code

Easily reuse existing .NET code

Visual Studio integration

Develop, debug, submit jobs, fine-tune performance

Scale-out automatically

Scale-out and parallelize U-SQL scripts automatically

Store unlimited data

Store virtually unlimited data in Azure Data Lake
Anatomy of a U-SQL query

10 log records by Duration (End time minus Start time). Sort rows in descending order of Duration.

**Query 1**

```
REFERENCE ASSEMBLY WebLogExtASM;

@rs =
EXTRACT
    UserID string,
    Start DateTime,
    End DateTime,
    Region string,
    SitesVisited string,
    PagesVisited string
FROM "swebhdfs://Logs/WebLogRecords.txt"
USING WebLogExtractor();

@result = SELECT UserID,
          (End.Subtract(Start)).TotalSeconds AS Duration
FROM @rs
ORDER BY Duration DESC FETCH 10;

OUTPUT @result TO "swebhdfs://Logs/Results/top10.txt"
USING Outputter.Tsv();
```
## U-SQL data types

<table>
<thead>
<tr>
<th>Category</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text</strong></td>
<td><code>char, char?</code>&lt;br&gt;<code>string</code></td>
</tr>
<tr>
<td><strong>Complex</strong></td>
<td><code>MAP&lt;K&gt;</code>&lt;br&gt;<code>ARRAY&lt;K,T&gt;</code></td>
</tr>
<tr>
<td><strong>Temporal</strong></td>
<td><code>DateTime, DateTime?</code></td>
</tr>
<tr>
<td><strong>Other</strong></td>
<td><code>bool, bool?</code>&lt;br&gt;<code>Guid, Guid?</code>&lt;br&gt;<code>Byte[]</code></td>
</tr>
</tbody>
</table>

**Note:** Nullable types have to be declared with a question mark `?`
Use ARRAY type to hold the list of pages visited

```usql
@rs1 =
SELECT 
    UserId,
    new ARRAY<string>(PagesVisited.Split( new [] { ';' } )) AS VisitedPagesArray 
FROM @rs;

@rs2 =
SELECT 
    UserId AS Users,
    VisitedPagesArray.Count AS VisitedPages 
FROM @rs1 
WHERE VisitedPagesArray.Count > 10;

OUTPUT @rs2 TO @"results.tsv" USING Outputters.Tsv();
```

**Desired Output**

<table>
<thead>
<tr>
<th>User ID</th>
<th>Visited Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>A$A892</td>
<td>12</td>
</tr>
<tr>
<td>HG54#A</td>
<td>29</td>
</tr>
<tr>
<td>YSD78@</td>
<td>14</td>
</tr>
<tr>
<td>JADI899</td>
<td>45</td>
</tr>
<tr>
<td>YCPB(%U)</td>
<td>30</td>
</tr>
<tr>
<td>HADS46$</td>
<td>18</td>
</tr>
<tr>
<td>MVDRY79%</td>
<td>12</td>
</tr>
<tr>
<td>TYUSPS67</td>
<td>14</td>
</tr>
</tbody>
</table>
Reading files with custom formats

Use built-in extractors to read CSV and TSV files, or create custom extractors for different formats

1. Implement IExtractor Interface

```csharp
using Microsoft.SCOPE.Interfaces;
public WebLogExtractor:IExtractor
{
    public override IEnumerable<IRow> Extract(...)
    {
        ...
        ...
    }
}
```

2. Upload and Register Assembly

```sql
CREATE ASSEMBLY WebLogExtAsm
FROM @"/WebLogExtAsm.dll"
WITH PERMISSION_SET = RESTRICTED;
CREATE EXTRACTOR WebLogExtractor
EXTERNAL NAME WebLogExtractor;
```

3. Reference the Assembly and Use

```sql
REFERENCE ASSEMBLY WebLogExtAsm;
// now just use it like a built-in extractor
SELECT * FROM @"swebhdfs://Logs/WebRecords.txt"
USING WebLogExtractor();
```
WebLogExtractor: Implementation

[Code snippet]

Custom extractor for WebLogRecords.txt file with this format
Aggregation functions

Can be extended with custom aggregation functions

Built-in aggregation functions
- AVG
- ARRAY_AGG
- COUNT
- FIRST
- LAST
- MAP_AGG
- MAX
- MIN
- STDEV
- SUM
- VAR

Query 3
Count users per region and the AVG, MAX, MIN and total duration
Create performance with TABLEs

**Query 4**

Improve performance of Query1!

```sql
CREATE TABLE LogRecordsTable(UserId int, Start DateTime, End Datetime, Region string
INDEX idx CLUSTERED (Region ASC) PARTITIONED BY HASH (Region));
```

**WebLogRecords.txt**

**INSERT INTO** LogRecordsTable
**SELECT** UserId, Start, End, Region
**FROM** @rs;

**@result** =
**SELECT** UserID, (End.Subtract(Start)).TotalSeconds **AS** Duration
**FROM** LogRecordsTable **ORDER BY** Duration **DESC** **FETCH 10**
**OUTPUT** @result **TO** "swebhdfs://Logs/Results/Top10.Tsv"
**USING** Outputters.Tsv();

**Azure Data Lake**

**Top10.Tsv**
Partitioned tables: Improve performance

CREATE TABLE LogRecordsTable (UserId int, Start DateTime, Region string, INDEX idx CLUSTERED (Region ASC) PARTITIONED BY HASH (Region));

INSERT INTO LogRecordsTable
SELECT UserId, Start, End, Region FROM @rs

During insertion records are hash distributed among the 3 extents based on value of ‘Region’ column

@rs = SELECT * FROM LogRecordsTable
WHERE Region == “en-gb”

Partition elimination
Table: Clustered indexes

Improve performance by eliminating ‘cross-extent shuffling’

```sql
@rs1 = SELECT Region, COUNT(*) AS Total FROM @rs GROUP BY Region;
@rs2 = SELECT TOP 100 Region, Total FROM @rs1 ORDER BY Total;
```
Table valued functions

- Functions that return a TABLE object
- Parameterized “view”—they are in-lined (parameterized lambdas)
- Can contain U-SQL expressions except OUTPUT and DDL/DML
- Benefits:
  - Eliminates unnecessary repetition of code
  - Enables sharing of code
  - Makes scripts more manageable

```sql
//create the TVF
DROP FUNCTION IF EXISTS SearchLogView;  -- DDL statement
CREATE FUNCTION WebLogView ()
RETURNS @result TABLE (
    UserId int, Start DateTime, End DateTime, Region string,
    SiteVisited string, PagesVisited string
) AS BEGIN
    @result =
    EXTRACT UserId int, Start DateTime, End DateTime, Region string,
    SiteVisited string, PagesVisited string
    FROM "swebhdfs://Logs/WebRecords.txt"
    USING WebLogExtractor();
    RETURN;
END;

//now use the TVF
@searchlog = WebLogView();
OUTPUT @searchlog TO "swebhdfs://Logs/WebRecords_copy.tsv"
USING Outputters.Tsv();
```
U-SQL extensibility
Extend U-SQL with C#/.NET

Built-in operators, function, aggregates

- User-defined operators (UDOs)
- User-defined functions (UDFs)
- User-defined aggregates (UDAGGs)
- C# expressions (in SELECT statements)
## Types of user-defined operators

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extractors</td>
<td>Extract</td>
</tr>
<tr>
<td>Processors</td>
<td>Process</td>
</tr>
<tr>
<td>Appliers</td>
<td>Apply</td>
</tr>
<tr>
<td>Combiners</td>
<td>Combine</td>
</tr>
<tr>
<td>Reducers</td>
<td>Reduce</td>
</tr>
<tr>
<td>Outputters</td>
<td>Output</td>
</tr>
</tbody>
</table>
Federated queries: Query data where it lives

Easily query data in multiple Azure data stores without moving it to a single store

Benefits

- Avoid moving large amounts of data across the network between stores
- Single view of data irrespective of physical location
- Minimize data proliferation issues caused by maintaining multiple copies
- Single query language for all data
- Each data store maintains its own sovereignty
- Design choices based on the need
Azure Portal integration

- Create a new Big Data Analytics Account
- Author U-SQL scripts
- Submit U-SQL jobs
- Cancel running jobs
- Provision users who can submit jobs
- Visualize usage stats (compute hours)
- Visualize job management chart
Visual Studio integration
What can you do with Visual Studio?

- Author U-SQL scripts (with C# code)
- Debug U-SQL and C# code
- Submit and cancel U-SQL Jobs
- Visualize physical plan of U-SQL query
- Visualize and replay progress of job
- Fine-tune query performance
- Create metadata objects
- Browse metadata catalog
“Data read” playback (video)

⚡ For performance tuning, identify bottlenecks and debugging, you can playback the job execution graph
ADL Analytics

Getting started

- Developing big data apps
- Works across cloud data
- Simplified management and admin
Get started

1. Log in to Azure
2. Create an ADLA account
3. Write and submit an ADLA job with U-SQL (or Hive/Pig)
4. The job reads and writes data from storage

ADLS
Azure Blobs
Azure DB
...
Azure Data Lake Pricing
ADLA billing

- Accounts are **FREE!**
- Pay for the compute resources you want for your **queries**
- Pay for **storage separately**

\[(\text{query\_minutes} \times \text{parallelism} \times \text{parallelism\_cost\_per\_minute}) + \text{per\_job\_charge}\]