**WDCloud:** An End to End System for Large-Scale Watershed Delineation on Cloud

*In Kee Kim, *Jacob Steele, +Anthony Castronova, *Jonathan Goodall, and *Marty Humphrey

*University of Virginia
+Utah State University
Watershed Delineation

• **Watershed Delineation:**
  - A starting point of many hydrological analyses.
  - Defining a watershed boundary for the area of interests.

• **Why Important?**
  - Defining the scope of modeling domain.
  - Impacting further analysis and modeling steps of hydrologic research.
Approaches for Large-Scale Watershed Delineation

• **Approaches:**
  - Commercial Desktop SWs (e.g. GIS tools).
  - Online Geo-Services (e.g. USGS – StreamStats).
  - Algorithms/Mechanisms from Research Community.

• **Limitations:**
  - Steep Learning Curve.
  - Requiring Significant Amount of Preprocessing.
  - Scalability and Performance for nation-scale watersheds.
  - Uncertainty of Execution (Watershed Delineation) Time.
The goals of this research is addressing:

1. The **Scalability Problem** of public dataset (NHD+)-based approach (Castronova and Goodall’s approach).
2. The **Performance Problem** of very large-scale watershed delineations (e.g. the Mississippi) using the recent advancement of computing technology (e.g. Cloud and MapReduce).
3. The **Predictability Problem** of watershed delineation using ML (e.g. Local Linear Regression).
Our Approach

1. Automated Catchment Search Mechanism Using NHD+

2. Performance Improvement for Computing a Large Number of Geometric Union:
   a. Data-Reuse
   b. Parallel-Union
   c. MapReduce

3. LLR (Local Linear Regression)-based Execution Time Estimation
Our Approach

1. Automated Catchment Search Mechanism Using NHD+.
   ➔ To address the Scalability Problem.
2. Performance Improvement for Computing a Large Number of Geometric Union:
   a. Data-Reuse
   b. Parallel-Union ➔ To address the Performance Problem.
   c. MapReduce
3. LLR (Local Linear Regression)-based Execution Time Estimation.
   ➔ To address the Predictability Problem.
Design of WDCloud

<table>
<thead>
<tr>
<th>WDCloud Component</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Web Portal for WDCloud** | - Provides UI (Bing Maps) to select target watershed coordinates.  
- Displays the final delineation results (as well as output files (KML)). |
| **NHD+ Dataset** | - Has a single NHD+ DB (SQL Server) by integrating 21 district NHD DBs. |
| **Automated Catchment Search Module** | - Collects relevant catchments in multiple NHD regions for the target watershed. |
| **Geometric Union Module** | - Performs geometric union operation to create the final watershed. |
| **Execution Time Estimator** | - Estimate duration for the given watershed delineation via LLR. |
| **Amazon Web Services** | - Various compute resources (e.g., VMs) and storage resources (e.g., Amazon S3) for WDCloud. |
Automated Catchment Search Module

• Automatically search and collect all relevant catchments in multiple NHD+ regions via *HydroSeq*, *TerminalPath*, and *DnHydroSeq*.

### Algorithm 1: Automated Catchment Search for Multiple Regions in NHD+

```python
Require: coord: coordinate for outlet of the target watershed
1: start_HUC_region ← get_regional_dataset(coord)
2: terminal_paths ← get_terminal_path_infos(start_HUC_region, coord)
3: catchments ← get_catchments(start_HUC_region, terminal_paths)
4: multi_region_hydroseqs ← get_multi_region_hydroseqs_info(start_HUC_region, terminal_paths)
5: if length(multi_region_hydroseqs) > 0 then
6: related_HUC_regions ← find_related_HUC_regions(multi_region_hydroseqs)
7: region_index ← 0
8: while region_index < length(related_HUC_regions) do
9: catchment_for_HUC_region ← get_catchments(related_HUC_regions[region_index], terminal_paths)
10: catchments.append(catchment_for_HUC_region)
11: region_index++
12: end while
13: end if
```

• Output: *Set of Catchments that forms the target watershed.*
# Performance Improvement Strategies

<table>
<thead>
<tr>
<th>Domain Specific</th>
<th>Strategy</th>
<th>Description</th>
<th># of Catchments</th>
<th># of VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-Reuse</td>
<td>For the “monster-scale” watersheds (e.g. the Mississippi).</td>
<td>Multi-HUC region case.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(approx. 1.1mil+)</td>
<td></td>
</tr>
<tr>
<td>System Specific</td>
<td>Parallel Union</td>
<td>Maximize the performance of single VM.</td>
<td>&lt; 25K</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>MapReduce</td>
<td>Maximize the performance of watershed delineation via Hadoop Cluster.</td>
<td>&gt;= 25K</td>
<td>&gt; 1</td>
</tr>
</tbody>
</table>
Performance Improvement – “Data-Reuse”

• **Key Idea:**
  - Pre-compute catchment unions for Monster-scale Watersheds. (not using a specific point for outlet).
  - Offline optimization to guarantee the performance of watershed delineations.

![Diagram showing NHD+ Region “A” and NHD+ Region “B+C” (Pre-computed)]
Performance Improvement – “Data-Reuse”

• Key Idea:
  - Pre-compute catchment unions for Monster-scale Watersheds. (not using a specific point for outlet).
  - Offline optimization to guarantee the performance of watershed delineations.
Performance Improvement – “Data-Reuse”

• **Key Idea:**
  - Pre-compute catchment unions for Monster-scale Watersheds. (not using a specific point for outlet).
  - Offline optimization to guarantee the performance of watershed delineations.

![Diagram showing only merging catchments in Region “A” (Green Area) with Target Watershed, NHD+ Region “A”, and NHD+ Region “B+C” (Pre-computed).]
Performance Improvement – “Data-Reuse”

- **Key Idea:**
  - Pre-compute catchment unions for Monster-scale Watersheds. (not using a specific point for outlet).
  - Offline optimization to guarantee the performance of watershed delineations.
Performance Improvement – “Parallel-Union”

• Key Idea:
  - Used for medium-size (less than 25K catchments) watersheds.
  - Designed to maximize a multi-core (up to 32 cores) single VM instance.
  - Watershed delineation can be parallelized via “Divide-and-Conquer” or “MapReduce Style” computation.

A collection of catchments for Target Watershed
Performance Improvement – “MapReduce”

• Key Idea:
  - “Hadoop version” of Parallel-Union.
  - Designed to maximize the performance (minimize the watershed execution time) via utilizing multiple numbers of VM instances.
  - Used for large-size (more than 25K catchments) watersheds.
Execution Time Estimation – LLR (Local Linear Regression)

• **Initial** Hypothesis:
  - Execution time for watershed delineation has a *somewhat linear relationship with IaaS/Application (Watershed Delineation Tool) specific parameters* (e.g. VM Type, # of Catchments)

• **Watershed Delineation Tool** has several pipeline steps that each pipeline step is related to:
  - Geometric Union (Polygon Processing)
  - Non-Geometric Union

• **Data Collection and Correlation Analysis**
  - Profiled 26 execution samples on 4 different Types of VMs on AWS.

<table>
<thead>
<tr>
<th></th>
<th># of Catchment</th>
<th>Type of VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non Geometric Union</td>
<td>0.0973 (negligible)</td>
<td>0.7089 (strong)</td>
</tr>
<tr>
<td>Geometric Union</td>
<td>0.6129 (moderate)</td>
<td>0.3223 (weak)</td>
</tr>
</tbody>
</table>

**Simple Linear Model → Cannot Produce Reliable Prediction**
Execution Time Estimation – LLR (Local Linear Regression)

“GLOBAL” LINEAR REGRESSION VS. “LOCAL” LINEAR REGRESSION

(a) Global Linear regression on m1.large (using all samples)  (b) Local Linear Regression on m1.large (Using three samples)

• Procedure of Local Linear Regression

1. Applying $k$NN to find a proper set $V(x_0)$ for prediction.
2. Creating simple Regression model based on $V(x_0)$
3. Making prediction for Job $x_0$ based on the Regression model

- # of Catchment
- Geographical Closeness
- Exec. Environment (VM)
Evaluation (1) – Performance Improvement

(1) Data-Reuse
(Monster Watershed)

Mississippi Watershed

<table>
<thead>
<tr>
<th>Comm. Desktop</th>
<th>Data Reuse</th>
<th>Speed Ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>10+ Hrs</td>
<td>5.5 min.</td>
<td>111x</td>
</tr>
</tbody>
</table>

4 Core i7 with 8G RAM

M1.xlarge Instance on AWS (4 vCPUs with 7.5G Ram)

(2) Parallel-Union
(# of catch. < 25K)

Norm. Execution Time

≈ 1200 sec.

3.9x speedup
(≈ 310 sec.)

(3) MapReduce
(# of catch. >= 25K)

MapReduce

Speed-Up (Baseline: Non-parallel)

Large-Scale Watersheds (# of Catchment)

11.8 min.
Evaluation – Execution Time Estimation (Overall)

- Measures 420 random coordinates.
  - (20 random coordinates for watershed outlet * 21 HUC regions in NHD+)
- Metrics:
  1) Prediction Accuracy
  2) MAPE (Mean Absolute Percentage Error)

\[
\text{Prediction Accuracy} = \begin{cases} 
\frac{T_{\text{actual}}}{T_{\text{predicted}}} & , T_{\text{predicted}} \geq T_{\text{actual}} \\
\frac{T_{\text{predicted}}}{T_{\text{actual}}} & , T_{\text{actual}} > T_{\text{predicted}} 
\end{cases}
\]

\[
\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{T_{\text{actual},i} - T_{\text{predicted},i}}{T_{\text{actual},i}} \right|
\]

Overall Results for Execution Time Estimation

<table>
<thead>
<tr>
<th></th>
<th>LLR Estimator</th>
<th>(Geo) kNN</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prediction Accuracy</td>
<td>85.6%</td>
<td>65.7%</td>
<td>42.8%</td>
</tr>
<tr>
<td>MAPE</td>
<td>0.19</td>
<td>0.93</td>
<td>1.97</td>
</tr>
</tbody>
</table>
Evaluation – Execution Time Estimation (Regional)

Prediction Accuracy

- LLR Predictor
- kNN
- mean

MAPE

- LLR Predictor
- kNN
- mean
Conclusions

• We have designed and implemented **WDCloud** on top of public cloud (AWS) to solve three limitations of existing approaches:
  1) Scalability → Automated Catchment Search Mechanism.
  2) Performance → Three Perf. Improvement Strategies.
  3) Predictability → Local Linear Regression.

• Evaluations of **WDCloud** on AWS:
  • Performance Improvement
    - 4x ~ 111x speed up (Parallel Union, MapReduce, Data Reuse)
  • Prediction Accuracy
    - 85.6% of prediction accuracy and 0.19 of MAPE.
Questions?

Thank you!
Support Slides (NHD+ Regions)