DCBench: a Benchmark Suite for Data Center Workloads

Zhen Jia

Institute of Computing Technology, Chinese Academy of Sciences

HVC tutorial in conjunction with The 19th IEEE International Symposium on High Performance Computer Architecture (HPCA 2013)
Content

• Background and Motivation
• DCBench
• Application scenarios for DCBench workloads
• Usage example
• Use cases
Data in the World

1986
Analog
2.62 billion GB
Digital
0.02 billion GB

2000
Analog Storage

2007
Analog
18.86 billion GB
Digital
276.12 billion GB

Data Never Sleeps

Data Is Created Every Minute!

http://www.domo.com/blog/2012/06/how-much-data-is-created-every-minute/?dkw=socf3
Data Centers in the World

Emerson December 2011
State-of-Practice Benchmark Suites

- SPEC CPU
- SPEC Web
- HPCC
- PARSEC
- TPCC
- Gridmix
- YCSB
Why a New Benchmark Suite

• No benchmark suite covers diversity of data center workloads

• State-of-art: CloudSuite
  – Only includes 6 applications according to its popularity
Why a New Benchmark Suite (Cont’)

- Memory Level Parallelism (MLP):
  Simultaneously outstanding cache misses

CloudSuite

our benchmark suite

MLP

CloudSuite
DCBench
Why a New Benchmark Suite (Cont’)

• Scale-out performance

Cloudsuite Data analysis benchmark

Working nodes

Speed up

DCBench

- sort
- grep
- wordcount
- svm
- kmeans
- fkmeans
- all-pairs
- Bayes
- HMM

HPCA 2013 | HVC Tutorial
## Why a New Benchmark Suite (Cont’)

<table>
<thead>
<tr>
<th>Diverse Programming Models</th>
<th>CloudSuite</th>
<th>HiBench</th>
<th>WL suite</th>
<th>MineBench</th>
<th>GridMix</th>
</tr>
</thead>
<tbody>
<tr>
<td>representative applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic operations</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>n</td>
<td>y</td>
</tr>
<tr>
<td>classification</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>clustering</td>
<td>n</td>
<td>y</td>
<td>n</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>recommendation</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>sequence learning</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>association rule mining</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>data warehouse operations</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>feature reduction</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>vector calculus</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>bioinformatics application</td>
<td>n</td>
<td>n</td>
<td>n</td>
<td>y</td>
<td>n</td>
</tr>
<tr>
<td>distributed</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>n</td>
<td>y</td>
</tr>
</tbody>
</table>
Content

- Background and Motivation
- DCBench
- Application scenarios for DCBench workloads
- Usage example
- Use cases
DCBench

• Goal

• Methodology

• Workloads
Target Class of Machines

Data centers
The Requirements of Benchmarks for Data Centers

Target Benchmarks

- Representative
- Diverse Programming Models
- Distributed
- State-of-art
DCBench

• Goal

• Methodology

• Workloads
Methodology

• To decide and rank main application domains according to a publicly available metric — e.g. page view and daily visitors

• To single out the main applications from main applications domains

• To cover different programming models
Methodology

• To decide and rank main application domains according to a publicly available metric—e.g. page view and daily visitors

• To single out the main applications from main applications domains

• To cover different programming models
Top Sites on the Web

More details in http://www.alex.com/topsites/global;0
Methodology

- To decide and rank main application domains according to a publicly available metric—e.g. page view and daily visitors
- To single out the main applications from main applications domains
- To cover different programming models
Algorithms in Top Sites: Search Engine

- Search Engine: 40%
- Social Network: 25%
- Electronic Commerce: 15%
- Media Streaming: 5%
- Others: 15%

Top Sites on The Web

Algorithms used in Search:
- Pagerank
- Graph mining
- Segmentation
- Feature Reduction
- Grep
- Statistical counting
- Vector calculation
- Sort
- Recommendation
- ….
Algorithms in Top Sites: Social Network

Algorithms used in Social Network:
- Recommendation
- Clustering
- Classification
- Graph mining
- Grep
- Feature Reduction
- Statistical counting
- Vector calculation
- Sort
- ……
Algorithms in Top Sites: Electronic Commerce

- Search Engine: 15%
- Social Network: 25%
- Electronic Commerce: 40%
- Media Streaming: 15%
- Others: 5%

Algorithms used in electronic commerce:
- Recommendation
- Associate rule mining
- Warehouse operation
- Clustering
- Classification
- Statistical counting
- Vector calculation
- ……
Main Algorithms in Data Centers

- Basic operation
- Classification
- Cluster
- Recommendation
- Association rule mining

- Segmentation
- Warehouse operation
- Feature reduction
- Vector calculate
- Graph mining

Data center algorithms
Methodology

• To decide and rank main application domains according to a publicly available metric – e.g. page view and daily visitors

• To single out the main applications from main applications domains

• To cover different programming models
Programming Models

- The same algorithm implemented with different programming models demonstrates varied performance results.

<table>
<thead>
<tr>
<th>Programming model</th>
<th>Data Set (KB)</th>
<th>Wall Time (second)</th>
<th>Processing data (KB/S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI</td>
<td>31420</td>
<td>658</td>
<td>47.75</td>
</tr>
<tr>
<td>MapReduce</td>
<td>31420</td>
<td>2165</td>
<td>14.52</td>
</tr>
</tbody>
</table>

3.3 times
DCBench

• Goal

• Methodology

• Workloads
DCBench workloads

• A benchmark suite for data center
  – Three kinds of workloads
    • Data analysis applications
    • Service
    • Interactive Real-time applications
  – Programming models
    • MapReduce
    • MPI
    • All-pairs
## Overview of DCBench

<table>
<thead>
<tr>
<th>Category</th>
<th>Workloads</th>
<th>Programming model</th>
<th>language</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic operation</td>
<td>Sort</td>
<td>MapReduce</td>
<td>Java</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>Wordcount</td>
<td>MapReduce</td>
<td>Java</td>
<td>Hadoop</td>
</tr>
<tr>
<td></td>
<td>Grep</td>
<td>MapReduce</td>
<td>Java</td>
<td>Hadoop</td>
</tr>
<tr>
<td>Classification</td>
<td>Naïve Bayes</td>
<td>MapReduce</td>
<td>Java</td>
<td>Mahout</td>
</tr>
<tr>
<td></td>
<td>Support Machine</td>
<td>Vector</td>
<td>Java</td>
<td>Implemented by ourself</td>
</tr>
<tr>
<td>Cluster</td>
<td>K-means</td>
<td>MapReduce</td>
<td>Java</td>
<td>Mahout</td>
</tr>
<tr>
<td></td>
<td>Fuzzy k-means</td>
<td>MapReduce</td>
<td>Java</td>
<td>Mahout</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPI</td>
<td>C++</td>
<td>IBM PML</td>
</tr>
<tr>
<td>Recommendation</td>
<td>Item based Collaborative Filtering</td>
<td>MapReduce</td>
<td>Java</td>
<td>Mahout</td>
</tr>
<tr>
<td>Association rule mining</td>
<td>Frequent growth pattern</td>
<td>MapReduce</td>
<td>Java</td>
<td>Mahout</td>
</tr>
<tr>
<td>Segmentation</td>
<td>Hidden model</td>
<td>MapReduce</td>
<td>Java</td>
<td>Implemented by ourself</td>
</tr>
</tbody>
</table>
## Overview of DCBench (Cont’)

<table>
<thead>
<tr>
<th>Category</th>
<th>Workloads</th>
<th>Programming model</th>
<th>language</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouse operation</td>
<td>Database operations</td>
<td>MapReduce</td>
<td>Java</td>
<td>Hive-bench</td>
</tr>
<tr>
<td>Feature reduction</td>
<td>Principal Component Analysis</td>
<td>MPI</td>
<td>C++</td>
<td>IBM PML</td>
</tr>
<tr>
<td></td>
<td>Kernel Principal Component Analysis</td>
<td>MPI</td>
<td>C++</td>
<td>IBM PML</td>
</tr>
<tr>
<td>Vector calculate</td>
<td>Paper similarity analysis</td>
<td>All-Pairs</td>
<td>C&amp;C++</td>
<td>Implemented by ourself</td>
</tr>
<tr>
<td>Graph mining</td>
<td>Breadth-first search</td>
<td>MPI</td>
<td>C++</td>
<td>Graph500</td>
</tr>
<tr>
<td></td>
<td>Pagerank</td>
<td>MapReduce</td>
<td>Java</td>
<td>Mahout</td>
</tr>
<tr>
<td>Service</td>
<td>Search engine</td>
<td>C/S</td>
<td>Java</td>
<td>Implemented by ourself</td>
</tr>
<tr>
<td></td>
<td>Auction</td>
<td>C/S</td>
<td>Java</td>
<td>Rubis</td>
</tr>
<tr>
<td>Interactive real-time application</td>
<td>Media streaming</td>
<td>C/S</td>
<td>Java</td>
<td>Cloudsuite</td>
</tr>
</tbody>
</table>
## Each Algorithm’s Application Scenarios

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Application Scenarios</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sort</td>
<td>Ranking the pages according to its importance (PageRank)</td>
</tr>
<tr>
<td></td>
<td>Pages sorting by its ID (Web storage in database)</td>
</tr>
<tr>
<td>Wordcount</td>
<td>Calculating the TF-IDF base information, such as term frequency</td>
</tr>
<tr>
<td></td>
<td>Obtain the user operations count to analysis their social behavior (in Wolfram Alpha)</td>
</tr>
<tr>
<td>Grep</td>
<td>Log analysis</td>
</tr>
<tr>
<td></td>
<td>Web information extraction</td>
</tr>
<tr>
<td></td>
<td>Fuzzy search</td>
</tr>
<tr>
<td>Naïve Bayes</td>
<td>Spam recognition (Spam Filtering with Naïve Bayes)</td>
</tr>
<tr>
<td></td>
<td>Bioinformatics (Naïve Bayesian Classifier for Rapid Assignment of RNA Sequences into the New Bacterial Taxonomy)</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>Classification (Question Classification)</td>
</tr>
<tr>
<td></td>
<td>Image Processing (Image annotation)</td>
</tr>
<tr>
<td></td>
<td>Text Categorization</td>
</tr>
</tbody>
</table>
## Each Algorithm’s Application Scenarios (Cont’)

| K-means                  | Image processing (Fast image segmentation)  
<table>
<thead>
<tr>
<th></th>
<th>High-resolution landform classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item-based Collaborative Filtering</td>
<td>Amazon recommender system</td>
</tr>
<tr>
<td>Hidden Markov model</td>
<td>Bioinformatics (Protein homology detection)</td>
</tr>
<tr>
<td></td>
<td>Speech recognition, Handwriting recognition</td>
</tr>
<tr>
<td></td>
<td>Word Segmentation</td>
</tr>
<tr>
<td>Frequent pattern growth</td>
<td>Market Analysis</td>
</tr>
<tr>
<td></td>
<td>Data mining in Business (identifying competitive suppliers in Supply Chain Management)</td>
</tr>
<tr>
<td></td>
<td>Intrusion detection</td>
</tr>
<tr>
<td></td>
<td>Query Recommendation</td>
</tr>
<tr>
<td>Warehouse operation</td>
<td>Taobao Yunti system</td>
</tr>
<tr>
<td></td>
<td>Facebook</td>
</tr>
<tr>
<td></td>
<td>Yahoo!</td>
</tr>
<tr>
<td>Principal Component Analysis</td>
<td>computer vision</td>
</tr>
<tr>
<td></td>
<td>pattern recognition</td>
</tr>
<tr>
<td></td>
<td>Face Representation and Recognition</td>
</tr>
</tbody>
</table>
Where Do Those Algorithms Exactly Used in Data Centers?

Here, let’s investigate mostly used applications in data centers

– The ubiquitous search engine
– Frequently used recommendation sub-systems
Algorithms in Search Engine

More details in [1]
# Representative Algorithms in Search Engine

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Role in the search engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>graph mining</td>
<td>crawl web page</td>
</tr>
<tr>
<td>Grep</td>
<td>abstracting content from HTML</td>
</tr>
<tr>
<td>segmentation</td>
<td>word segmentation</td>
</tr>
<tr>
<td>pagerank</td>
<td>compute the page rank value</td>
</tr>
<tr>
<td>Word counting</td>
<td>word frequency count</td>
</tr>
<tr>
<td>vector calculation</td>
<td>document matching</td>
</tr>
<tr>
<td>sort</td>
<td>document sorting</td>
</tr>
</tbody>
</table>
Algorithms in Recommendation Sub-systems

- Web structure mining
  - Crawler
  - Link analysis

- Web content mining
  - Web pages
    - preparation
    - text representation
  - Human labeling
  - learner
    - models
  - Structural pages
    - information extraction

- Web usage mining
  - preparation
  - user behavior log
    - user behavior representation
  - association rule mining
    - user classification
    - user clustering
  - collaborative filtering
    - recommendation process
  - user information extraction
  - user information collection
# Representative Algorithms in Recommendation Sub-systems

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Role in the recommendation sub-systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>classify web pages/user behavior</td>
</tr>
<tr>
<td>Frequent pattern growth</td>
<td>user log mining</td>
</tr>
<tr>
<td>Hidden markov model</td>
<td>information extraction</td>
</tr>
<tr>
<td>Clustering/similarity analysis</td>
<td>clustering web pages/user behavior</td>
</tr>
<tr>
<td>Collaborative filtering</td>
<td>recommendation</td>
</tr>
<tr>
<td>Feature reduction</td>
<td>text representation/user behavior representation</td>
</tr>
<tr>
<td>Graph mining</td>
<td>web link analysis</td>
</tr>
</tbody>
</table>
Content

• Background and Motivation
• DCBench
• Application scenarios for DCBench workloads
• Usage example
• Use cases
Basic Operations

• Algorithms:
  – Sort
  – Wordcount
  – Grep

• Application scenarios:
  – Ranking the pages according to its importance
  – Calculating the TF-IDF
  – Log analysis
  – Web information extraction
  – Fuzzy search
Classification

• Algorithms :
  – Naïve bayes
  – Support Vector Machine

• Application scenarios:
  – Image annotation
  – Data Mining
  – Text Categorization
Clustering Algorithm

• Algorithms:
  – k-means, fuzzy k-means
    • Grouping a set of objects. Objects in the same group are more similar to each other

• Application scenarios:
  – Image processing
  – High-resolution landform classification
Recommendation

• Algorithms:
  – Item-Based Collaborative Filtering
    • Find the similarity between items using data information, and recommend the items to users according to the similarities

• Application scenario:
  – Amazon recommender
Vector Calculation

• Algorithm:
  • vector similarity calculation

• Application scenarios:
  • Similarity analysis
  • Redundancy Elimination
PageRank

• Algorithms:
  – PageRank
    • a link analysis algorithm, which assigns a numerical weighting to web pages

• Application scenarios:
  – Google Search

\[ PR(A) = PR(B) + PR(C) + PR(D) \]
Feature Reduction

• Algorithms:
  – Principal Component Analysis (PCA) and Kernel Principal Component Analysis (KPCA)
    • Mapping the original high-dimension data onto a lower-dimensional space

• Application scenarios:
  – computer vision
  – pattern recognition
Graph Mining

• Algorithms:
  – Breadth-first search
    • begins at a root node and visits each of neighbor nodes in turn.

• Application scenario:
  – Search engine
  – Social network
Association Rule Mining

• Algorithms:
  • Frequent pattern growth
    — Find the relationship between items by building frequent pattern tree

• Application scenarios:
  • Market Analysis
  • Intrusion detection
Segmentation

• Algorithms:
  – Hidden Markov Model
    • Use the model to decide whether to segment

• Application scenarios:
  – Speech recognition
  – Handwriting recognition
Data Warehouse Operation

• Algorithms:
  – Hive based warehouse operation
    • SQL statements

• Application scenarios:
  – Facebook
  – Yahoo!
  – Taobao
Search Engine

• Service:
  – Nutch based search
  – Use Real trace to drive the search

• Similar websites:
  – Google
  – Yahoo!
  – Bing
Electronic Business

• Service:
  – Rubis

• Similar websites:
  – eBay
  – Amazon
  – Taobao
Media Streaming

• Application:
  – Darwin based application

• Similar website:
  – YouTube
  – hulu
  – Youku
Content

• Background and Motivation
• DCBench
• Application scenarios for DCBench workloads
• Usage example
• Use cases
Example 1

Q: How do you run naïve bayes?

Application process:

- Sample Data
- Trainer
- Model
- Results
- Data to be classified

Data to be classified
Example 1

Q: How do you run naïve bayes?
A: You should use the following command to prepare data

```
$HADOOP_HOME/bin/hadoop jar TextProduce.jar bayes-input file-100G 20 75000000 5
```

```
[root@hw084 TextProduce]# $HADOOP_HOME/bin/hadoop jar TextProduce.jar bayes-input file-100G 20 75000000 5
13/02/04 21:04:12 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.
****hdfs://172.18.11.84:9000/user/root/bayes-input/typepro
13/02/04 21:04:12 INFO input.TextInputFormat: Total input paths to process : 1
13/02/04 21:04:12 INFO util.NativeCodeLoader: Loaded the native-hadoop library
13/02/04 21:04:12 WARN snappy.LoadSnappy: Snappy native library not loaded
13/02/04 21:04:13 INFO mapred.JobClient: Running job: job_201302041725_0008
13/02/04 21:04:14 INFO mapred.JobClient: map 0% reduce 0%
13/02/04 21:04:33 INFO mapred.JobClient: map 100% reduce 0%
```
Training & Classification

A: run the benchmark using following command

Training :

${MAHOUT_HOME}/bin/mahout trainclassifier -i /user/root/bayes-out-1G -o /user/root/bayes-out-1G-mf-model -mf 100 -ms 10

Classification:

${MAHOUT_HOME}/bin/mahout testclassifier -m /user/root/bayes-out-1G-mf-model -d /user/root/file-100G
More Details in Data Generation

• Talk 2: A Benchmark Suite for Big Data Systems
• 10:30am to 11:40am
Example 2

• How to configure and run the search benchmark in $Search/exp/run-test.sh

#--write your workload here-----------------------------------#
report search.throughput.real.head: 100000
Fixed: 1000
Fixed: 100
Fixed: 100
Fixed: 100

Fixed part
Turning part
Fixed part
How to Configure Search

Application Name

Workload transforming functions & parameters

The annotation of this workload

search.throughput.real.head:100000-fixed:100

head:100000 - first 100,000 queries in real query list
fixed:100 - the query rate (100 requests per second), Uniform distribution

Format: “function1(:args)-function2(:args) …”
## Workload Transforming Function

<table>
<thead>
<tr>
<th>Function name</th>
<th>parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>head</td>
<td>$Total: $start</td>
<td>Get qs and ts from the sequence number of $start, and the total entry number of qs and ts is $Total, e.g. search.#anno.head:100:0@cf@req. If $start is 0 then it can be left out, e.g. search.#anno.head:100@cf@req.</td>
</tr>
<tr>
<td>uniq</td>
<td>NULL</td>
<td>Get the unique query terms out of qs e.g. search.#anno.uniq@cf@req.</td>
</tr>
<tr>
<td>random</td>
<td>$Total</td>
<td>Randomly get query terms from qs and the total number of queried terms is $Total, e.g. search.#anno.random:1000@cf@req.</td>
</tr>
<tr>
<td>shuffle</td>
<td>NULL</td>
<td>Shuffle the terms in qs, e.g. search.#anno.shuffle@cf@req.</td>
</tr>
<tr>
<td>hot</td>
<td>NULL</td>
<td>Sort the qs according to the frequency of terms’ occurrence, e.g. search.#anno.hot@cf@req.</td>
</tr>
<tr>
<td>lens</td>
<td>NULL</td>
<td>Sort the qs according to terms’ length.</td>
</tr>
<tr>
<td>blockreq</td>
<td>$Blocksize:$repeatCount</td>
<td>Repeat every $Blocksize terms in qs $RepeatCount times. e.g. search.#anno.blockreq:10:2@cf@req.</td>
</tr>
</tbody>
</table>
## Workload Transforming Function (Cont’)

<table>
<thead>
<tr>
<th>Function name</th>
<th>Parameters</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fixed</strong></td>
<td>$Rate$</td>
<td>Generate ts and set the query rate to be $Rate$ queries per second. e.g. search.#anno.fixed:20@cf@req</td>
</tr>
<tr>
<td><strong>burst</strong></td>
<td>$Rate:$K</td>
<td>Generate ts and let ts be $i<em>K</em>K/Rate$, where $i=1...len(qs)$ e.g. search.#anno.burst:20:2@cf@req</td>
</tr>
<tr>
<td><strong>scale</strong></td>
<td>$Rate$</td>
<td>Compress or amplify original ts by setting the query rate to be $Rate$ queries per second. e.g. search.#anno.scale:20@cf@req</td>
</tr>
<tr>
<td><strong>poisson</strong></td>
<td>$Rate$</td>
<td>Generate ts and make the query rate variation fit poisson distribution, and set the average rate to be $Rate$ queries per second, e.g. search.#anno.poisson:40@cf@req</td>
</tr>
<tr>
<td><strong>ratestep</strong></td>
<td>$Init:$step:$K$</td>
<td>Generate ts and set the initial query rate to be $Init$. The rate will increase for ($K-1$) times. Each time it will increase the value of $step$. Finally, it will be stable at the rate of “$Init + step * (K-1)” e.g. search.#anno.ratestep:20:5:20@cf@req</td>
</tr>
</tbody>
</table>
A More Complicated Example

Format: “function1(:args)-function2(:args) …”

Select 500,000 requests

the reach time of the requests follow poisson distribution, and the $\mu$ of the poisson is 100
Add Your Own Transforming Function

In `Search/searcher/searcher.py` trans_search_req_stream function()

Define your own function by using `def`

```python
def blockrep(ts, qs, bs, k):
    return None, list_sum(qs[bs*i:bs*(i+1)] * k for i in range(len(qs)/bs))

def slicestep(ts, qs, init, st, k=None):
    if k == None: k = (len(qs) - init)/st
    if init + st*k > len(qs): raise Exception('slicestep: not so many query')
    return None, list_sum([qs[j] for j in range(0, len(qs), init+i*st)] for i in range(k))
```
Run the Search Benchmark

user@localhost exp$ make test

Logs in file  **nohup.out**

1 + report search.htca.head:10000-fixed:40@s1i2@reqs-SoGou
2 + for i in '*$'
3 + make log/search.htca.head:10000-fixed:40@s1i2@reqs-SoGou/exp-report
4 make[1]: Entering directory `~/local/Search/exp'
5 job.py ans42:aa@gd47:../base pkill -9 java -/wait
6 [('gd47', 0)]
UI

• After running the benchmark you can visit:
  http://$yourserver:9090/nutch-1.1/
Content

• Background and Motivation
• DCBench
• Application scenarios for DCBench workloads
• Usage example
• Use cases
Use Case 1

Search workloads analysis (Real V.S. Synthetic)

- $T_{\text{response}} = T_{\text{queue}} + T_{\text{service}}$, response time and queue length have a linear relationship
- Rate variation can make the queue become longer
- More details in [2]
Use Case 2

- Storage system research in data center

- Got the I/O trace of our benchmark suites by using **blktrace**
  - File system optimization
  - Disk’s caching strategies
Use Case 3

• Memory system research in data center
  – Recommendation Benchmarks according to Memory Level Parallelism (MLP)
Reference


Open Discussion

Where do you think our benchmark should go?

What’s your suggestions for us?

Question?