ODYS: A Massively-Parallel Search Engine Using a DB-IR Tightly-Integrated Parallel DBMS

June 3rd, 2013

Kyu-Young Whang
KAIST Distinguished Professor/ACM Fellow/IEEE Fellow
Computer Science Department, KAIST

To be presented at ACM SIGMOD 2013, NY

Copyright © 2013 Kyu-Young Whang et al.

Contents

- Introduction
- DB-IR Integration
- ODYS Massively-Parallel Search Engine
- Performance Model
- Performance Evaluation
- Conclusions

Copyright © 2013 Kyu-Young Whang et al.

Web Search Engines

- A representative large-scale system, which handles billions of queries per day for a petabyte-scale database of tens of billions of Web pages [Dea09, Kun13, Nie10]

- Commercial Web Search engines (e.g., Google, Yahoo!)
  - Being implemented based on a scalable distributed file system (DFS such as GFS, HDFS) using a large number of commodity PCs

Copyright © 2013 Kyu-Young Whang et al.

Functionality of NoSQL Systems

- Distributed file systems (DFS), key-value stores (so-called "NoSQL" systems)
  - They have very simple and primitive functionality
  - They do not provide database functionality such as SQL, schemas, indexes, or query optimization
  - Developers need to implement high-level functionality by using low-level primitive functions

- Parallel processing frameworks such as MapReduce and Hadoop
  - They are known to be suitable for performing extract-transform-load (ETL) tasks or complex data analysis
  - But, they are not suitable for query processing on large-scale data because they are designed for batch processing and scanning of the whole data [Sto10]
  - Commercial search engines use them primarily for data loading or indexing instead of query processing

Copyright © 2013 Kyu-Young Whang et al.

High-level Functionality

- High-level functionality such as SQL, schemas, or indexes that are provided by the DBMS allows developers to implement queries that are used in search engines easily
  - providing a higher expressive power than primitive functions in key-value stores
  - facilitating easy (and much less error-prone) application development and maintenance

➔ Thus, there have been a lot of research efforts to support SQL even in NoSQL systems (e.g., Pig[Ols08], Hive[Thu09])

Copyright © 2013 Kyu-Young Whang et al.

Advantage of High-Level Functionality in a Search Engine

- Example schemas and SQL statements
  - Typical Queries:

    Query (b): Find the Web pages that contain the word "Obama"

    Query (c) (d): Find the Web pages that contain the word "Obama" from the site having siteID = 6000

Attribute Name	Attribute Type	Description
pageId	integer	Page identifier
siteId	integer	Site identifier
siteIdText	text	Site identifier
title	text	Page title
URL	varchar	Page URL
content	text	Page content

(a) *pageInfo* relation.

```
SELECT p.pageId
FROM pageInfo p
WHERE MATCH(p.content, "Obama")>0;
```

(b) SQL statement for keyword search

```
SELECT p.pageId
FROM pageInfo p
WHERE MATCH(p.content, "Obama")>0
AND p.siteId = 6000;
```

(c) SQL statement for *site-limited search.*

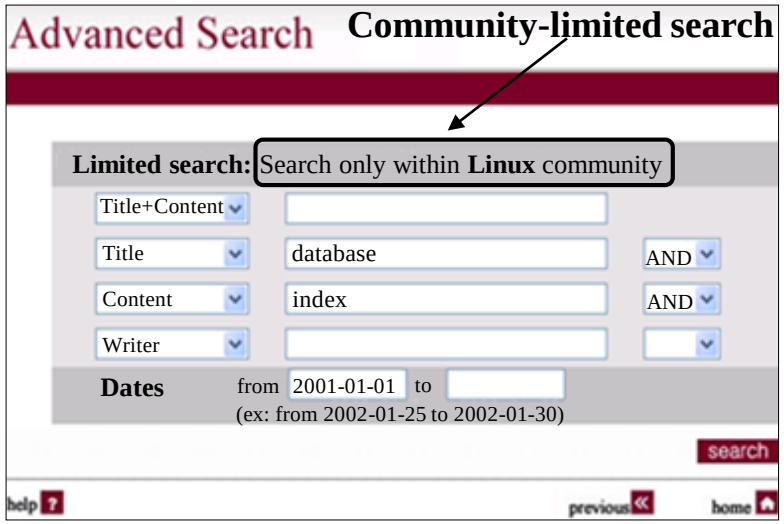
```
SELECT p.pageId
FROM pageInfo p
WHERE MATCH(p.content, "Obama")>0 AND MATCH(p.siteIdText, "6000")>0;
```

(d) An optimized version of SQL statement for *site-limited search.*

Note: Queries (c) and (d), which are a site-limited search, represent a DB-IR integrated query

Copyright © 2013 Kyu-Young Whang et al.

- Advanced search with multiple search fields — an on-line discussion board



(a) Advanced search with multiple fields.

```
SELECT p.pageId
FROM pageInfo p
WHERE MATCH(p.title, "database") > 0
AND MATCH(p.content, "index") > 0
AND MATCH(p.communityIdtext, "3") > 0
AND p.reg_date>="2001-01-01";
```

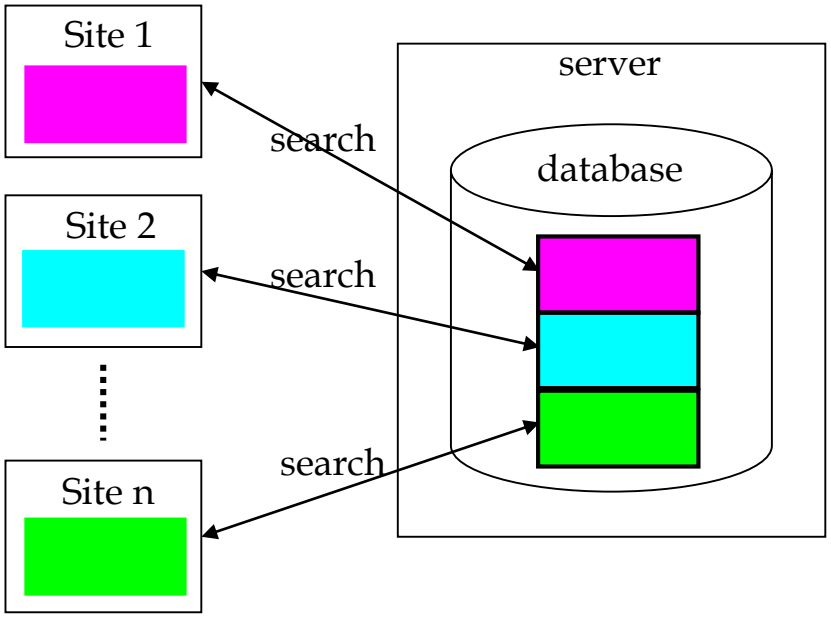
(b) An SQL statement for an advanced search using attribute embedding.

Note: Query (b) represents a DB-IR integrated query

Copyright © 2013 Kyu-Young Whang et al.

Site-limited Search

- Limiting the scope of a query to the set of web pages collected from a specific site



- Requiring DB-IR integration
  - Having both keyword and attribute conditions
  - Example: Find the web pages that contain the keyword "Obama" from www.whitehouse.gov

a keyword condition on text data

an attribute condition on structured data

Copyright © 2013 Kyu-Young Whang et al.

- Naïve Implementation
  - To find the record of the web page containing the word “Obama”
  - To access the records and select those whose siteId = 6000

  ⇒ Very bad in performance
  (Records are scattered all over the database causing excessive random accesses)

- Solutions
  - DB-IR tight integration

Copyright © 2013 Kyu-Young Whang et al.

Parallel DBMSs

- Parallel DBMSs could be considered an alternative to a large-scale search engine
  - having rich functionality such as SQL, schemas, indexes, and query optimization
  - providing parallel processing capability
  - having higher scalability and performance than traditional single node DBMSs

- Stonebreaker et al.[Sto10] argue that parallel DBMSs
  - are (linearly) scalable to handle large-scale data and query loads
  - can easily service multiple users for database systems with multi-petabytes of data

- However, parallel DBMSs have been considered as not having enough performance and scalability to be used **as a large-scale search engine** [Abo09, Dea04], one outstanding reason being the **lack of efficient information retrieval (IR) functionality**

Copyright © 2013 Kyu-Young Whang et al.

Our Contributions

- We show that we can construct a commercial-level massively-parallel search engine using a parallel DBMS, which to date has not be considered practical
  - Shared-nothing architecture with masters and slaves
  - Commercial-level scalability and efficiency by using a DB-IR tight-integrated DBMS
  - Higher-level functionality including SQL, schemas, and indexes
  - Each slave (DBMS) being capable of indexing 1 million Web pages

- We propose an analytic and experimental performance model (simply, a *hybrid model*) that estimates the performance of the proposed architecture and validate the accuracy of the model
  - Analytic (queuing) model : for estimating the **master and network** time (3.57 ~ 7.72%)
  - Experimental model : for estimating the **slave max** time (92.28 ~ 96.43%)

  ➔ We argue that this model can accurately estimate the performance of a massively-parallel search engine using the experimental results obtained from a small-scale one

  ➔ The hybrid model is helpful in realistically estimating the performance of a system by using limited resources without actually building a large-scale system

Copyright © 2013 Kyu-Young Whang et al.

- By using the performance model, we demonstrate that the proposed architecture is capable of handling commercial-level data and query loads with a rather small number of machines
  - The proposed architecture is expected to handle 1 billion queries/day (81 queries/sec) for 30 billion Web pages with an average query response time of
    - 194 ms with 43,472 nodes
    - 148 ms with 86,944 nodes

Copyright © 2013 Kyu-Young Whang et al.

DB-IR Integration

- Integration of DBMS with IR features has been studied actively as the need of handling unstructured data (e.g., text) as well as structured data is rapidly increasing

- DB and IR have been parallel universes [Cha05][Wei07]

	Database Systems	Information Retrieval
Canonical application	business/accounting	libraries
Data type	structured data (numbers, short strings)	unstructured data (text documents)
Foundation	algebra / logic	probability / statistics
Search paradigm	exact queries, sets/bags of results	vague queries, (ranked) lists of results

- But, many recent applications require integration of structured data and text data (e.g., CRM, E-commerce, etc.) [Wei07]

- It is preferable to support new data types (e.g., text) as the "first-class citizens" within the DBMS architecture [Lowell Report 2003]

Copyright © 2013 Kyu-Young Whang et al.

An Example: Digital Library

- Find papers about "spatial join" that are published after "1995"

Papers

title	authors	abstract	publication_year
Spatial Join Processing Using Corner Transformation	Song, J., Whang, K., Lee, Y., and Kim, S.	Spatial join finds pairs of spatial objects having a specific spatial relationship in spatial database systems. …	1999
…	…	…	…

```
SELECT    *
FROM      Papers
WHERE     ( Match(title, "spatial join") > 0 OR
            Match(abstract, "spatial join") > 0) AND
            publication_year > 1995;
```

query on text data

query on structured data

Copyright © 2013 Kyu-Young Whang et al.

Possible Approaches [Wha05]

- Loose-Coupling Architecture
  - Provides IR features as user defined types and functions outside of the DBMS engine (e.g., Oracle Cartridge and IBM Extender)
  - Poor performance: access paths are long; concurrency control and recovery in fine granularity are hard to perform;

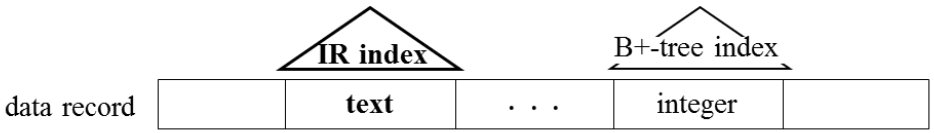
- Tight-Coupling Architecture ← *our approach*
  - IR features are implemented directly into the core of the DBMS engine (e.g., Odysseus [Wha02, Wha05, Wha13] and MySQL[Len04]) †
  - Good performance: access paths are short; concurrency control and recovery can be done in fine granularity; no extra inter-process communication overhead is incurred

  ➔ Tight coupling method is appropriate for a large-scale system to efficiently handle a large amount of data and high query loads

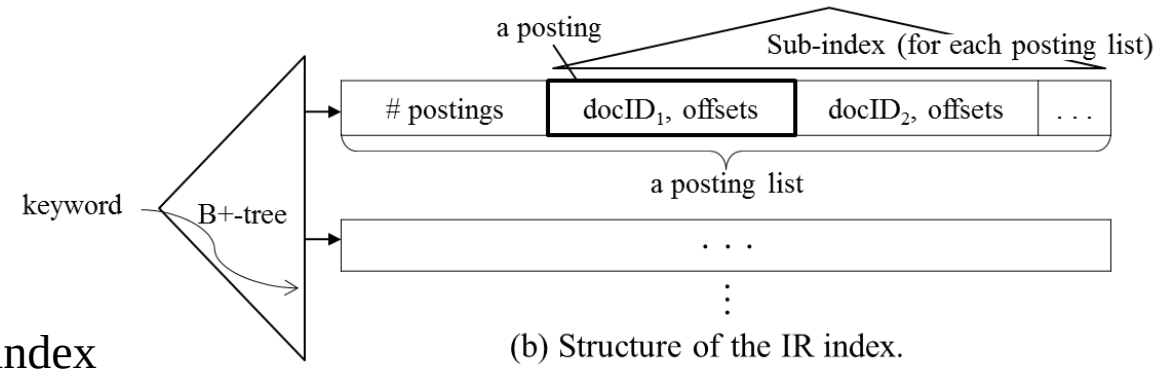
Copyright © 2013 Kyu-Young Whang et al.

Structure of the IR Index (U.S. Patented, 2002) [Wha02]

- IR index is (automatically) constructed for a column having the text type, consisting of
  - B+-tree index: for keywords, each keyword pointing to a *posting list*
  - Posting list: # of postings + postings for the keyword
  - Posting: document identifier (docID) + location information where the keyword appears (offset)
  - Sub-index



(a) IR index embedding.



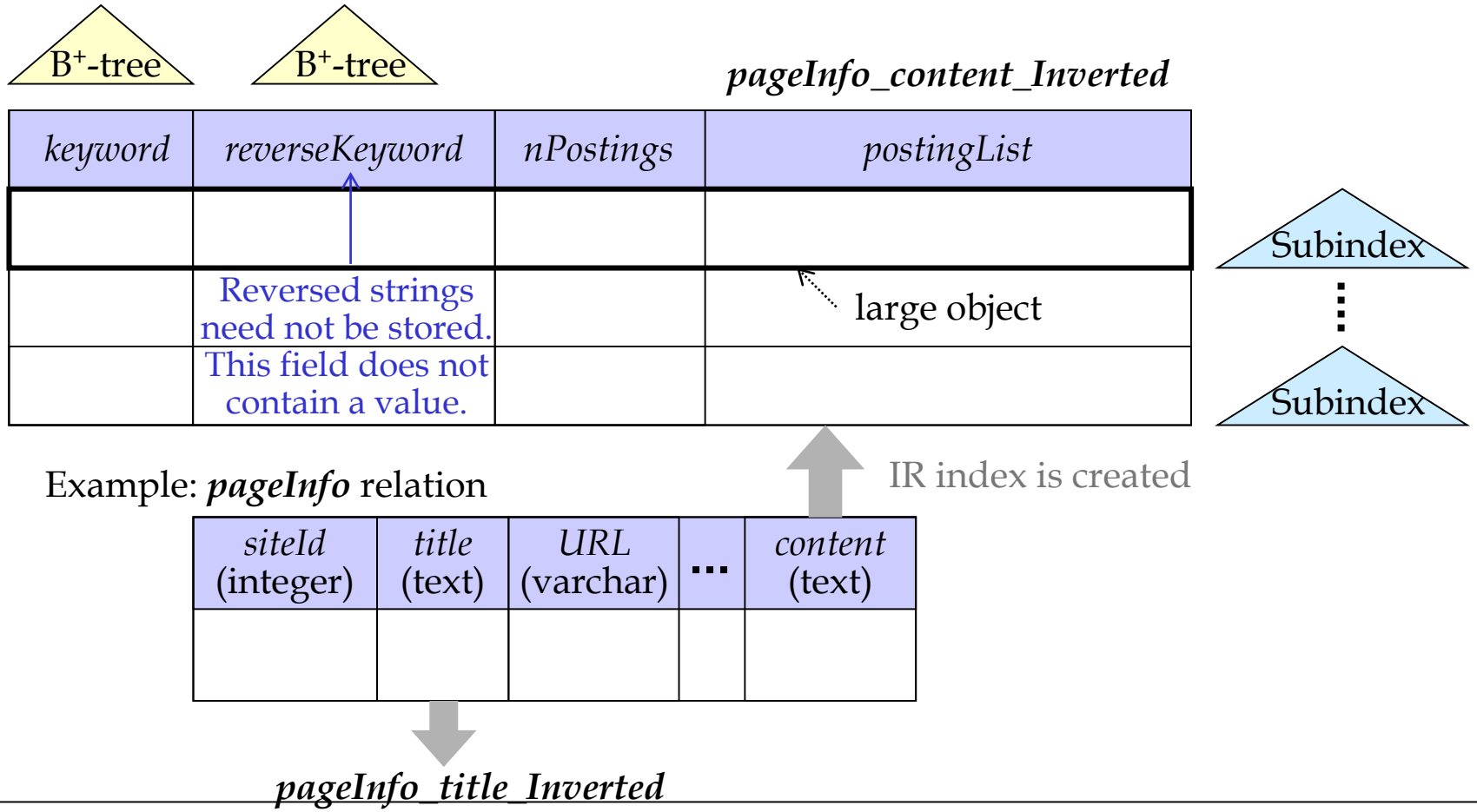
(b) Structure of the IR index.

- Sub-index
  - To index postings in each posting list
  - To allow quick finding of the location of a specific posting having a given docID within a posting list
  - To allow posting skipping and fast query processing

Copyright © 2013 Kyu-Young Whang et al.

Implementation of the IR Index

- IR index: created as a relation for each attribute of the TEXT type (the relation's name is "<*table name*>_<*attr name*>_Inverted")



Copyright © 2013 Kyu-Young Whang et al.

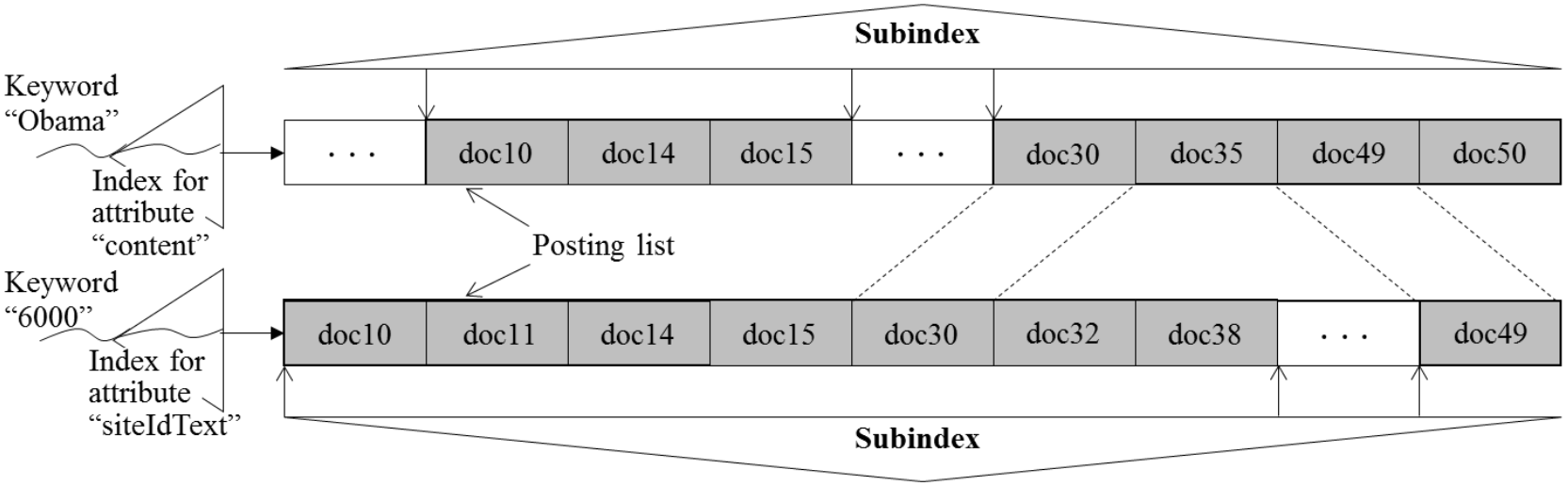
Compression of the IR index

- Compression: done primarily on the posting lists of the IR index
- Compression ratio: approximately 60%
  - The posting lists are compressed to 44% of their original size, but the subindexes are not compressed; thus, the overall compression ratio is **60**%
- Query performance: improved by approximately 20%
  - Reduced disk I/O's due to compressed IR index

Copyright © 2013 Kyu-Young Whang et al.

DB-IR Query Processing Using the IR Index

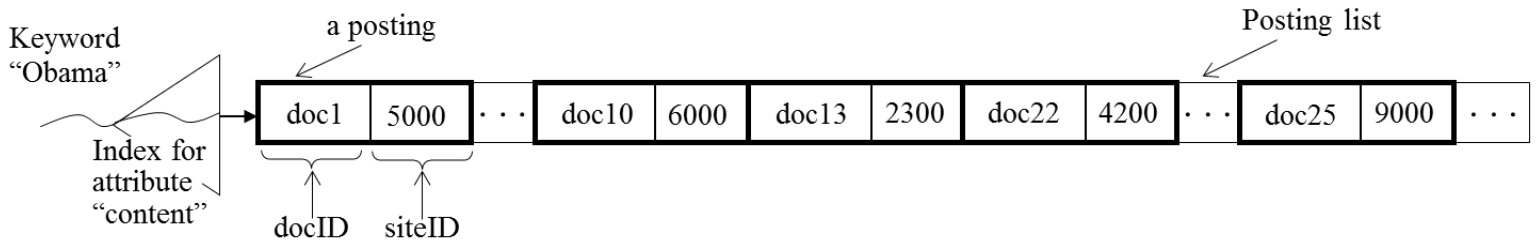
- ***IR Index Join*** [Wha03][Guo03][Hal03][Wha05]
  - Supporting fast query processing for multiple-keyword queries
  - Nontext attributes are redundantly stored as text attributes as well
  - ***Posting Skipping***: an optimization technique for IR Index Join
    - Using subindexes, the exact parts of posting lists that need to be merged can be identified



Copyright © 2013 Kyu-Young Whang et al.

- ***Attribute Embedding*** [Wha03][Wha05]
  - ***Attribute Embedding***: other attribute values of the record are embedded in the postings of another attribute
  - DB-IR integrated queries can be efficiently processed by embedding attribute values of the structured data in the postings of a text attribute
  - Attribute embedding can be specified through schema definition

    – Example: Create Table *pageInfo* (*siteID* integer, *content* **text(embedded_attributes(***siteID***))**, …);
    Note: ***siteID*** (of type integer) is embedded in the postings of the attribute ***content*** (of type text)



Copyright © 2013 Kyu-Young Whang et al.

Odysseus Object-Relational DBMS [WLL+05]

- Being developed at KAIST for over 23 years
- An earlier version of this technology played a vital role in starting up NaverCom Co. (currently, NHN Co.) (Founding CEO: Haejin Lee) in 1997-2000, which is the number one portal in Korea
- Winning the **Best Demonstration Award** at the IEEE 21st Int'l Conf. on Data Engineering (ICDE), Tokyo, Japan, Apr. 5-8, 2005
- Tightly coupling IR (U.S. patented) features as well as spatial database features
- Being a DBMS and, at the same time, a search engine
  - Providing concurrency control and recovery (coarse granularity and fine granularity)
  - Providing IR performance comparable to or better than those of commercial search engines
  - Indexing up to 100 million web pages per node
  - Allowing immediate updates
- Being a DBMS and, at the same time, a GIS engine
- Having many commercial applications
- Consisting of approximately 450,000 lines of C/C++ (high precision) codes

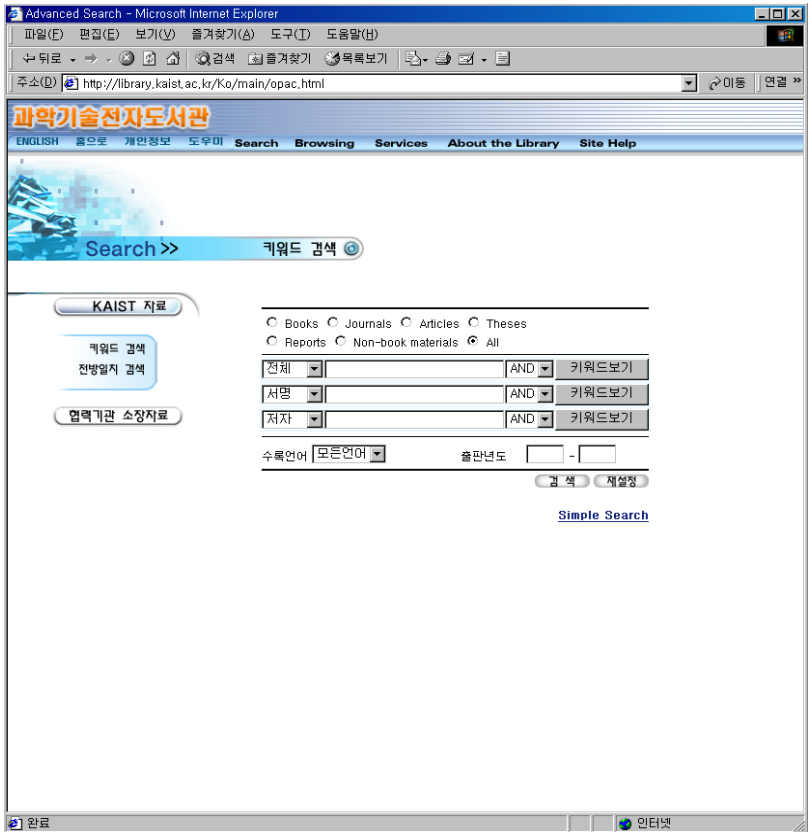
Copyright © 2013 Kyu-Young Whang et al.

수상 실적

- 한국공학상 (2012. 12): 대통령상/교과부/한국과학재단
- 2006년 우수연구성과 50선 (2006. 8): 한국과학재단
- 한국의 대표적 기초연구성과 30선 (2005.10): 한국과학재단
- 제 15회 과학기술우수논문상 (2005.5): 한국과학기술단체총연합회
- 최우수 시스템시연 논문상 (2005.4): IEEE Int'l Conf. on Data Engineering
- 20세기 100대 기술상 (1999.12) : 과학기술부/서울경제신문
- 정보문화상 기술상 (1999. 6): 정보통신부/한국정보문화센터
- 이달의 과학자상 (1998. 5): 과학기술부/한국과학재단
- 멀티미디어인상 기술 대상 (1997): 한국멀티미디어협회

Example1: A DBMS and, at the same time, a search engine

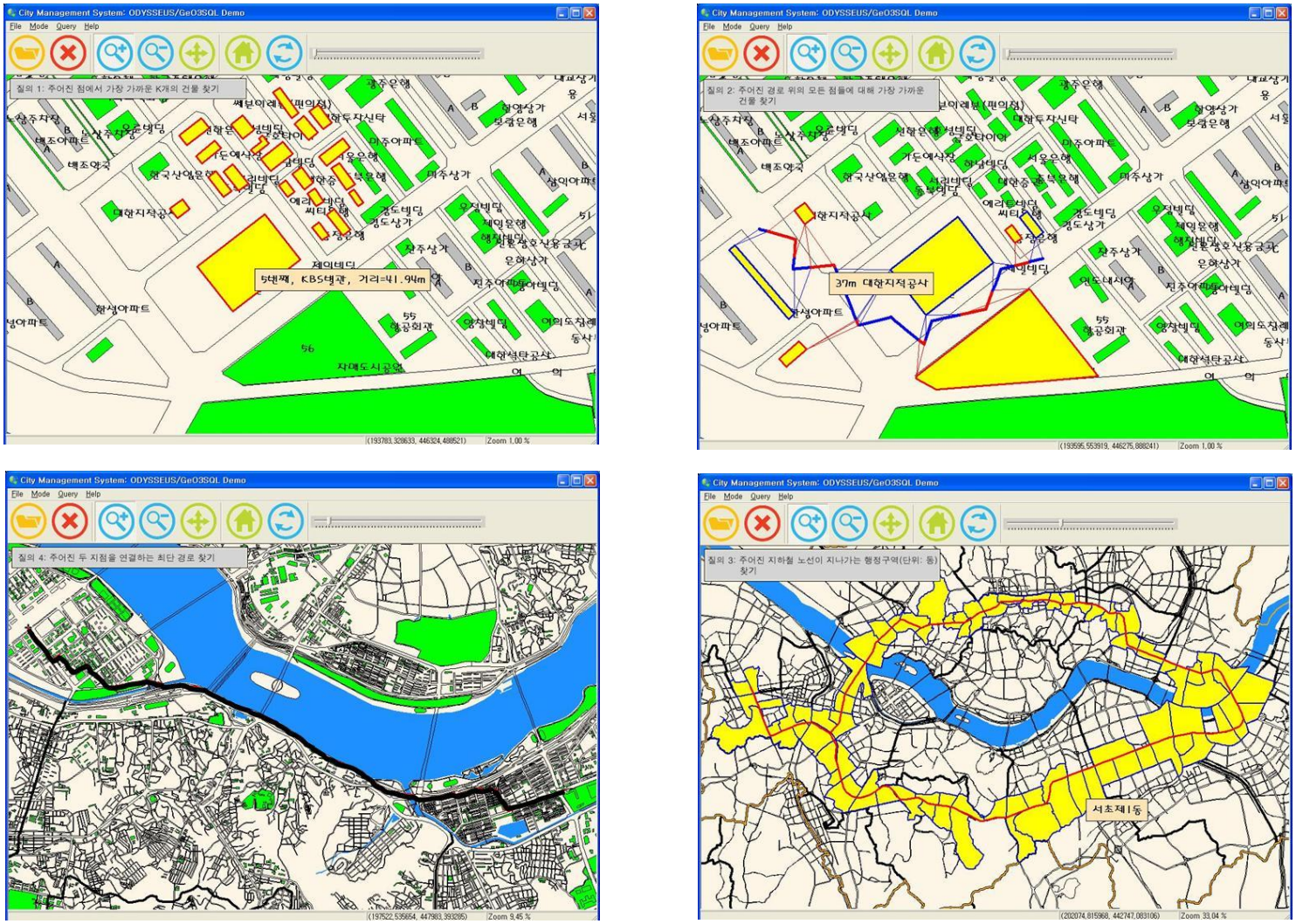
<ODYS, a Large-Scale Parallel Search Engine, powered by ODYSSEUS>



<KAIST Digital Library Search Engine (1998 ~ 2004) >

Copyright © 2013 Kyu-Young Whang et al.

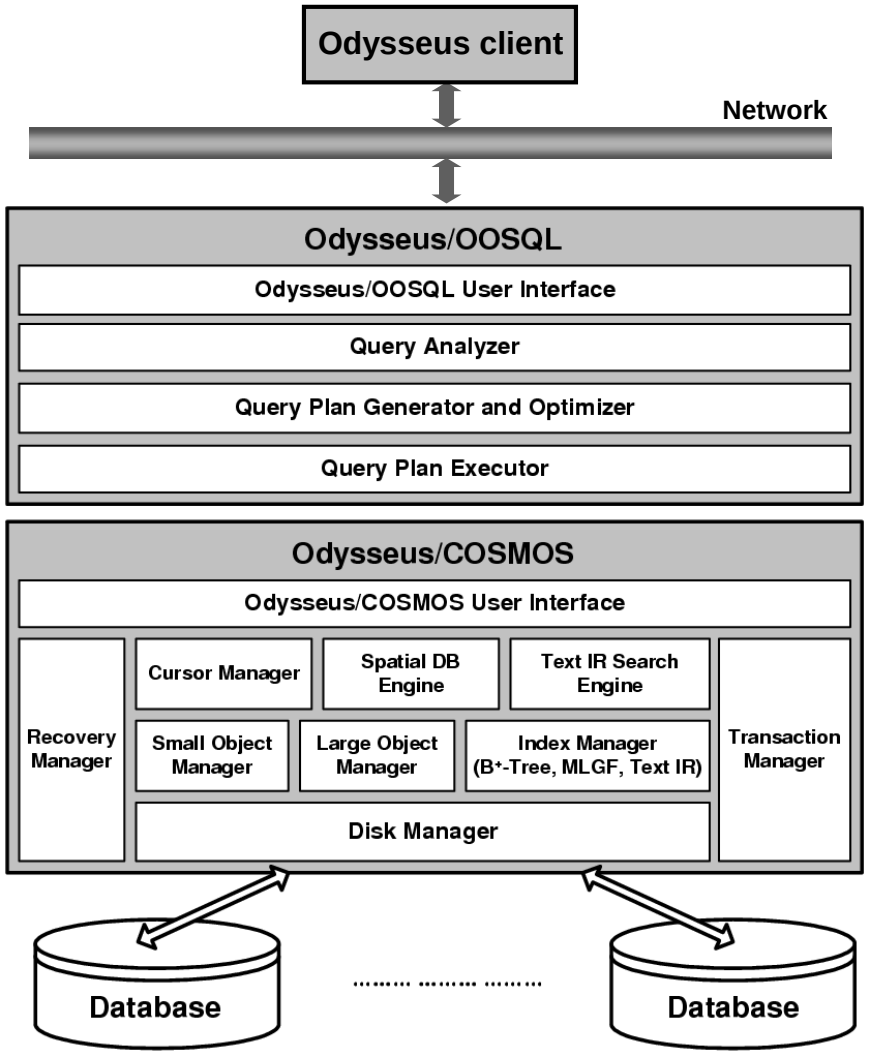
Example2: A DBMS and, at the same time, a GIS engine (a spatial DBMS)



<A Geographical Information System(GIS) Implemented Using ODYSSEUS>

Copyright © 2013 Kyu-Young Whang et al.

- Architecture of Odysseus
  - Client/server architecture
  - The Odysseus server consists of
    Odysseus/COSMOS : a storage system
    Odysseus/OOSQL : a query processor



Copyright © 2013 Kyu-Young Whang et al.

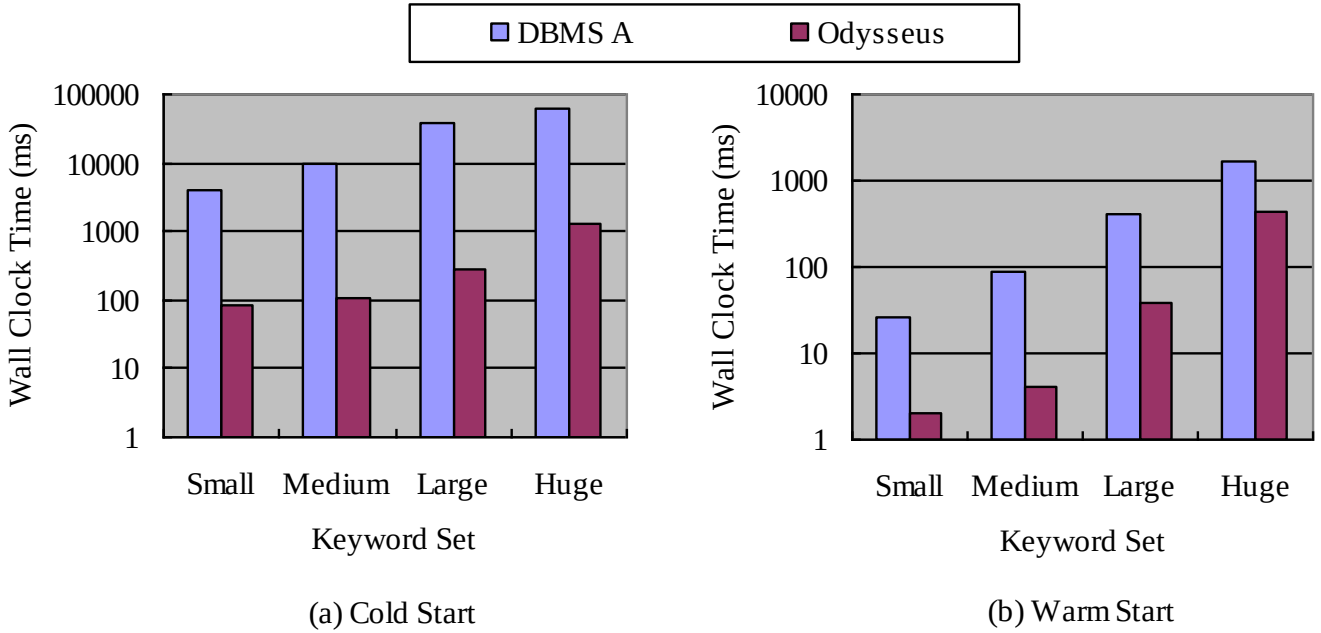
Performance Comparisons with DBMSs

- Setting: a SUN Blade 2000 workstation (900MHz CPU), a T3+ disk array
- DBMSs: Odysseus (*tight-coupling* architecture),
  DBMS A: a widely-used DBMS (*loose-coupling* architecture)
- Data: 15 million web pages (approximately 60 GBytes)
- Schema: *siteIdText*, *title*, *content* [1] attributes are of type 'text';
  other attributes are of type integer or varchar

[1] The size of *content* is limited to 8 KBytes.

Copyright © 2013 Kyu-Young Whang et al.

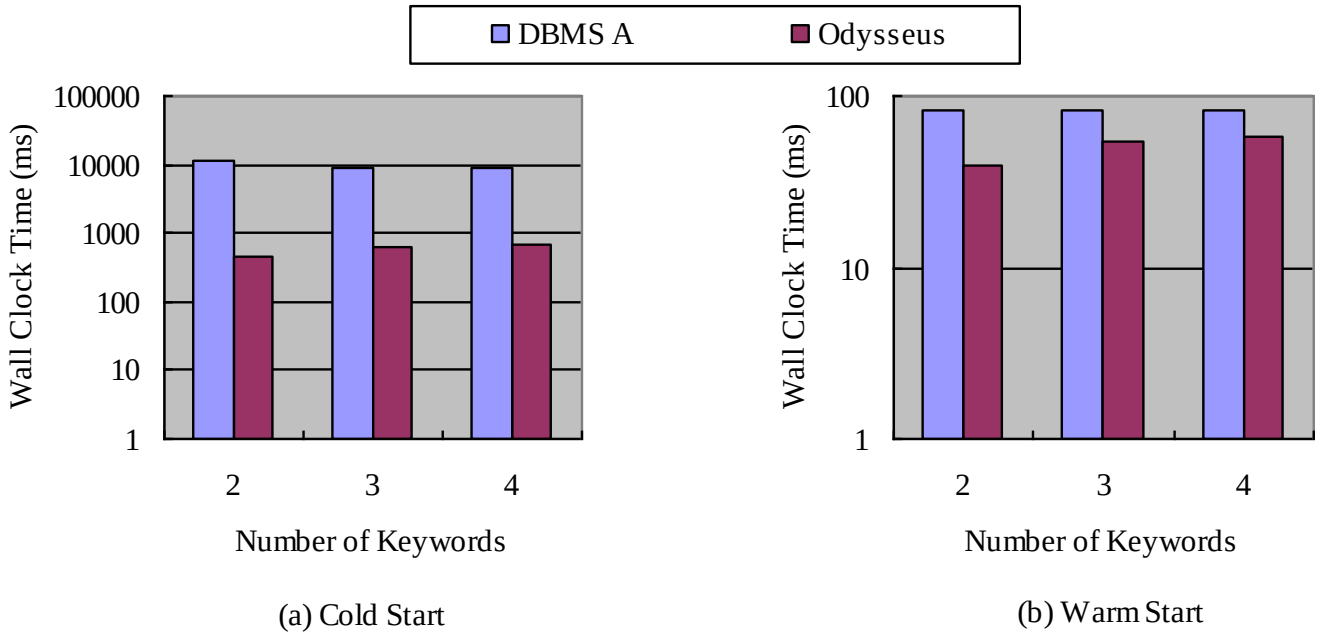
Results for Single-Keyword Queries



- Odysseus outperforms DBMS A by 47.3~133.6 times at cold start and by 4.0~21.6 times at warm start
- This result demonstrates the superiority of the tight-coupling architecture over the loose-coupling architecture

Copyright © 2013 Kyu-Young Whang et al.

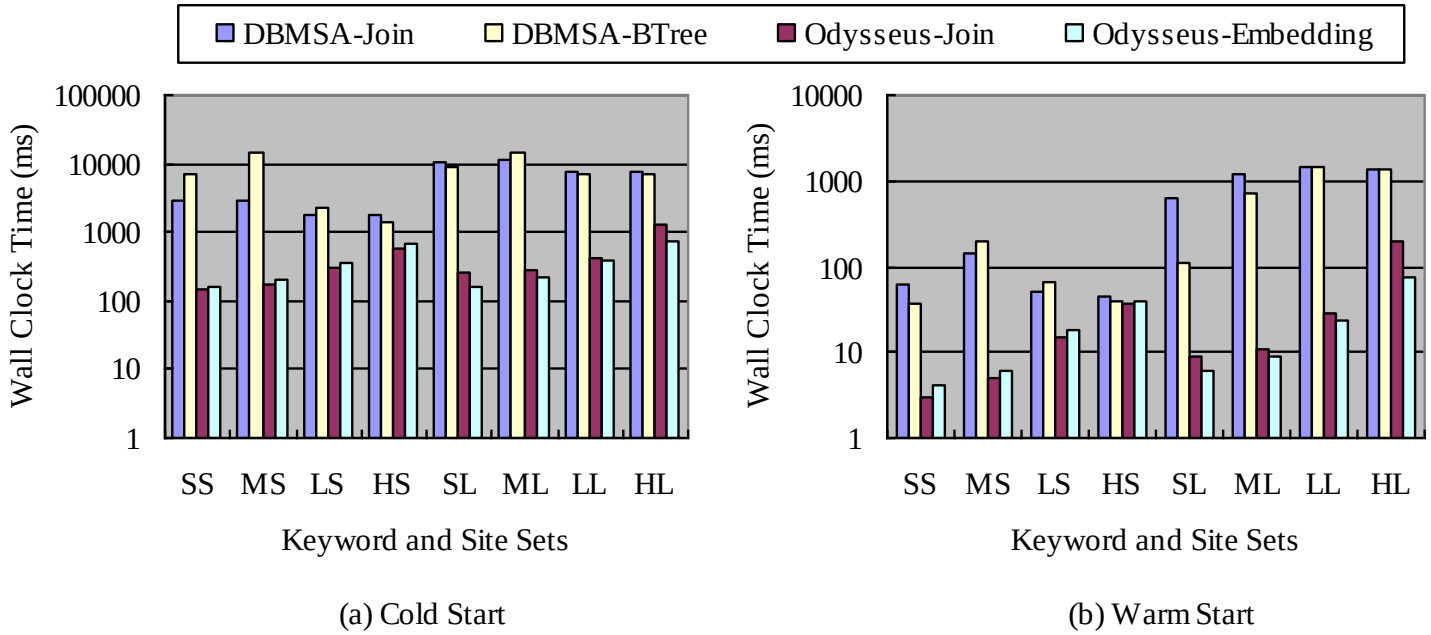
Results for Multiple-Keyword Queries



- Odysseus outperforms DBMS A by 13.2~24.8 times at cold start and by 1.4~2.1 times at warm start

Copyright © 2013 Kyu-Young Whang et al.

Results for Site-limited Search



- Odysseus-Join (IR index join) and Odysseus-Embedding (attribute embedding)
- Both methods significantly outperform DBMS A

Copyright © 2013 Kyu-Young Whang et al.

ODYS Search Engine

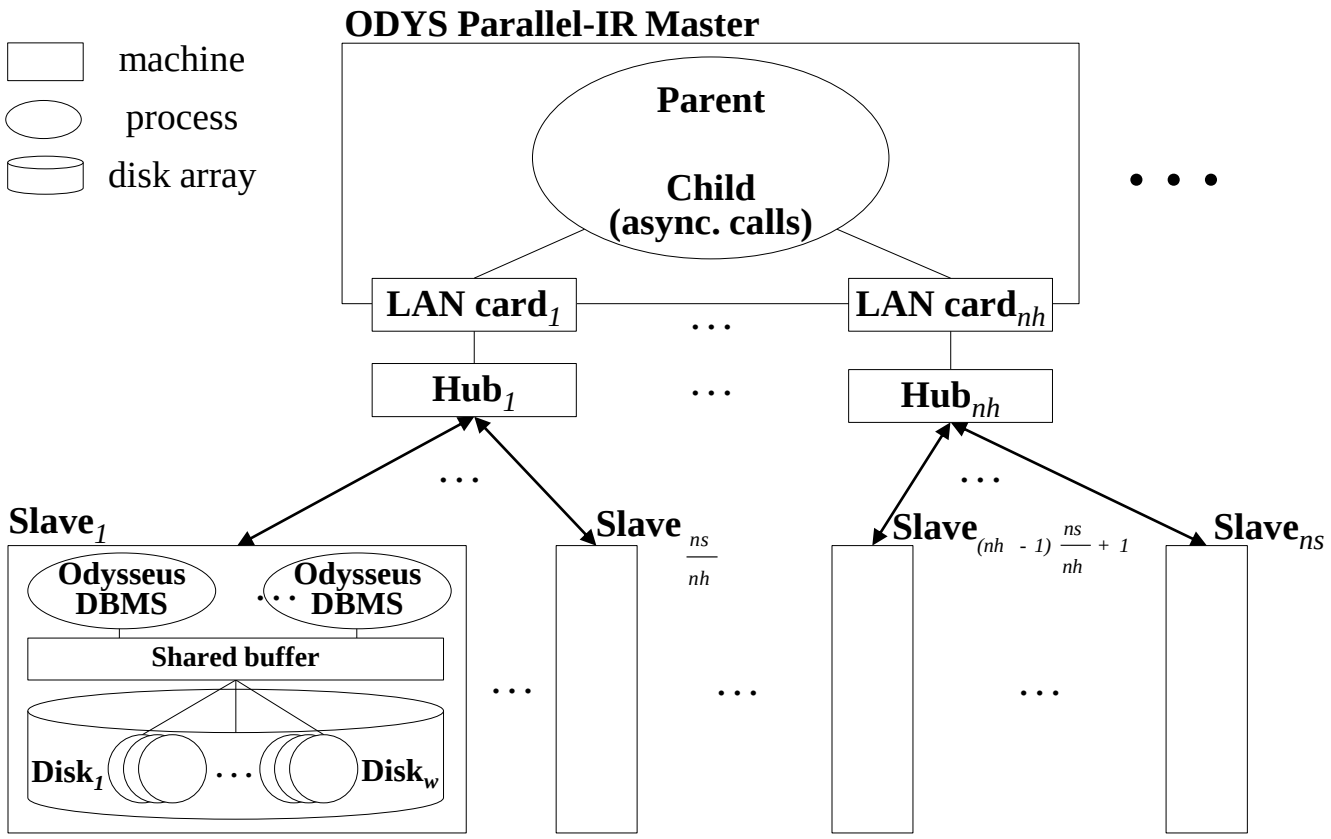
- A massively-parallel search engine using a DB-IR tightly-integrated parallel DBMS
- Efficiency
  - DB-IR tight integration
- Scalability
  - Shared-nothing architecture

Copyright © 2013 Kyu-Young Whang et al.

- High-level functionality
  - SQL (especially, DB-IR integrated queries)
    - Easy implementation of query interfaces by translating keyword queries into SQL queries
    - Selection, aggregation, limited join (where only one table is partitioned), etc.
  - Schemas
    - Activation of advanced query processing methods (i.e., attribute embedding or IR index join) by controlling the schema
  - Indexes
    - B+-tree indexes for structured data and IR indexes for unstructured data (i.e., text)
  - Limited transactions/consistency
    - Updates in a single machine with ACIDity
    - Good enough for large-scale applications such as on-line discussion (bulletin) boards or other SNS applications

Copyright © 2013 Kyu-Young Whang et al.

Architecture



Copyright © 2013 Kyu-Young Whang et al.

- Masters (ODYS Parallel-IR Master) †
  - Storing metadata(catalogs) such as global database schema, slaves' IP addresses, and slaves' database paths (i.e., the location of the disk device storing each slave database)

- Slaves (Odysseus Object-Relational DBMS) ‡
  - Shared-nothing architecture
  - Storing crawled Web pages and their IR indexes in a disk array
    - Entire set of Web pages is partitioned horizontally (i.e., by documents)

- Network †‡
  - The master and the slaves are connected by a gigabit network hub, and they communicate by using an asynchronous remote procedure call (RPC)

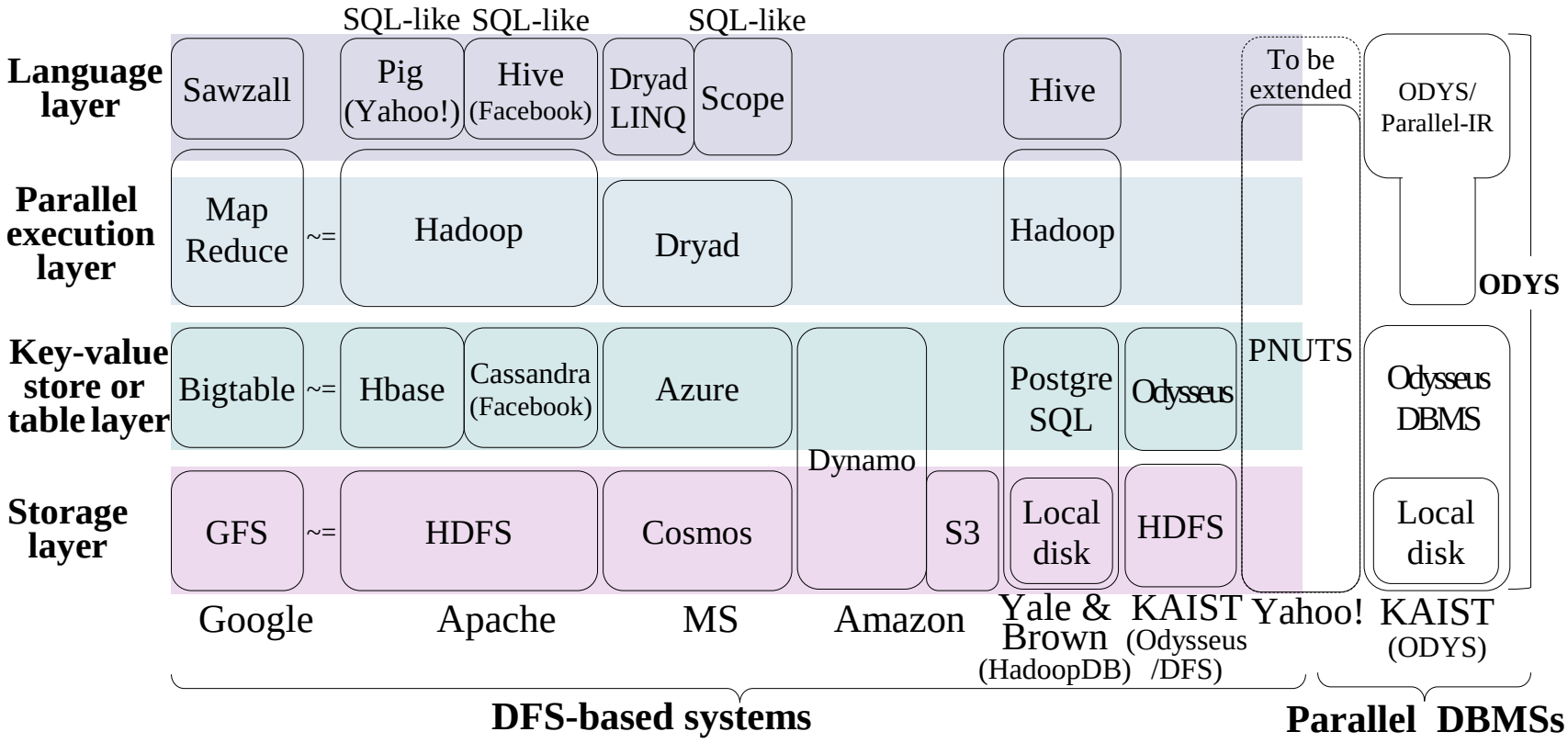
† The ODYS Parallel-IR Master consists of 58,000 lines of C and C++ code

‡ The Odysseus DBMS (slave) consists of 450,000 lines of C and C++ code

†‡ We use socket-based RPC consisting of 17,000 lines of C, C++, and Python code developed by the authors

Copyright © 2013 Kyu-Young Whang et al.

Map of ODYS and Other Parallel Processing Systems



Copyright © 2013 Kyu-Young Whang et al.

Performance Model – an Outline

- Hybrid (i.e., analytic and experimental) performance model
  - **Analytic:** master and network (Master CPUs, master memory buses, and network hubs)
    - We estimate the processing time of each component by using a queuing model
    - Even if the estimation error were sizable, it could not affect the overall performance in a significant way since the overall performance largely depends on the performance of the slave time (i.e., 92.28% ~ 96.43%)
  - **Experimental :** slave (slave CPUs, slave memory buses, and slave disk I/O)
    - They work in parallel, and the overall slave time is bounded by the maximum slave time
    - We estimate the processing time of the slaves by using an experimental (quasi measured) method; we call it the *partitioning method*
    - We measure the processing time of slaves at semi-cold start to obtain a lower-bounding performance
    - We can be assured that the estimated performance of slaves is very close to the actual measurement since the estimation is directly derived from the measurement

➔ Our performance model using a small-scale (e.g., 5-node) reference system is expected to quite accurately predict the performance of a large-scale (e.g., 300-node) system

Copyright © 2013 Kyu-Young Whang et al.

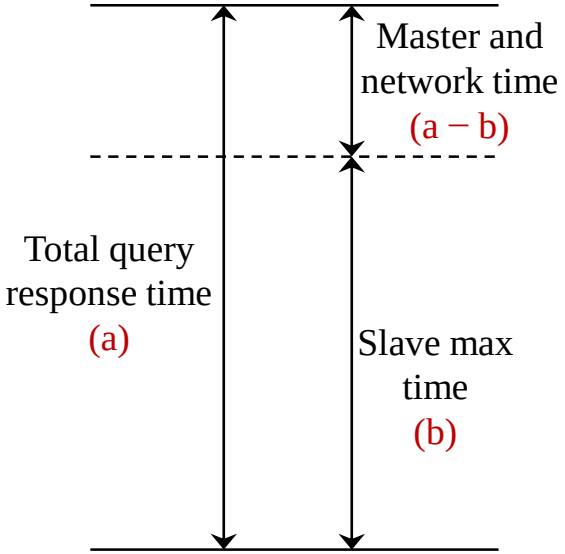
- Semi-cold start
  - A query is executed in the circumstance where the internal nodes of the IR indexes (which normally fit in main memory) are resident in main memory while the leaf nodes (which normally are larger than available main memory), posting lists, and the data (i.e., crawled Web pages) are resident in disk
  - We use a buffer of only 12 Mbytes sufficient for containing the internal nodes (occupying 11.5 Mbytes) of the IR index for each slave
  - Typical commercial search engines process queries at warm start by storing the entire (or a large part of) indexes and data in a massive-scale main memory. This helps significantly reduce the query response time

Copyright © 2013 Kyu-Young Whang et al.

Validation of Performance Model

- Estimation: Using a reference system (1 master, 5 slaves), we build the performance estimation model
  - Master and network (by an analytic queuing model)
    - Measure the parameters of the model with a 1-slave (1 master, 1 slave) system at a slowest speed (< 1million queries/day)
  - Slave (by an experimental model)
    - Measure slave max time with the reference system
    - 1 master, 5 slaves (5 slaves ×60 times = 300 data points)
    - Estimate slave max time for an 1 to 10-slave system by the *partitioning method*

- Experiments: Using a 1 to 10-slave system (1 master, 1 to 10 slaves), we obtain experimental results and compare them with the estimated results
  - Measure the total query response time: (a)
  - Measure slave max time: (b)
  - Calculate master and network time: (a) − (b)



Copyright © 2013 Kyu-Young Whang et al.

Query Model

- We use three types of search conditions: single-keyword query, multiple-keyword query, and limited-search query
- We consider top-10, top-50, and top-1000 queries
- Query load normalization
  - To simplify the queuing model, we normalize a query of a specific type into one equivalent query type: the single-keyword top-10 query (unit query)
  - Example (for master CPU):
    1 single-keyword top-1000 query = 1.79 single-keyword top-10 queries

Copyright © 2013 Kyu-Young Whang et al.

- We calculate the *weight* for each type of queries for each system component C (i.e., master CPU, master memory bus, and network) from the measurement
- We obtain a *weighted arrival rate (λ')* of queries for a system component *C* using a given query mix and the weights of query types
- Example: System component *C* = master CPU

< System component *C* time (ms) >

search condition \ *k*	Top-10	Top-50	Top-1000
All query types	25.014	25.536	44.740

< Weights of query types $w_c(k)$>

search condition \ *k*	Top-10	Top-50	Top-1000
All query types	1.0 query	1.02 queries	1.79 queries

< Query mix *qmr*(*sct*, *k*)>

search condition \ *k*	Top-10 (45%)	Top-50 (45%)	Top-1000 (10%)
Single keyword (98%)	44.1%	44.1%	9.8%
Multiple keyword (1%)	0.45%	0.45%	0.1%
Limited search (1%)	0.45%	0.45%	0.1%

$$\lambda_C' = \lambda_C \sum_{k \in \text{top-}k \text{ types}} \left(w_C(k) \left(\sum_{sct \in \text{search condition types}} qmr(sct, k) \right) \right)$$

$$= \lambda_C \times 1.088$$

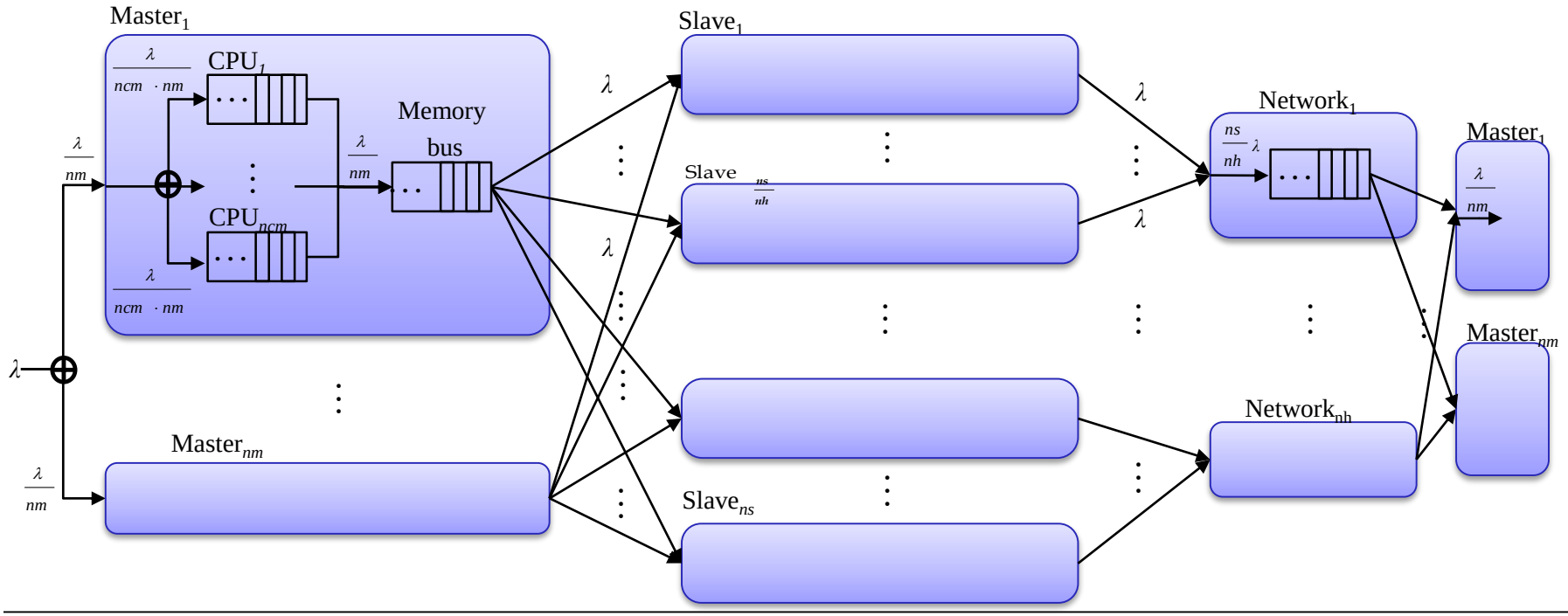
Copyright © 2013 Kyu-Young Whang et al.

Queuing Model (Master + Network)

- Arrival rate (λ)

Master CPU	Master main memory bus	Hub (network)
$\frac{\lambda'}{ncm \quad nm}$	$\frac{\lambda'}{nm}$	$\frac{ns}{nh}\lambda'$

λ: the weighted arrival rate of queries
nm: the number of master nodes
ncm: the number of CPUs per master
nh: the number of network hubs



Copyright © 2013 Kyu-Young Whang et al.

Model Parameters Measured

- Parameters of the queuing model measured

	Parameters	Values
Master CPU	$T_{parent\text{-}proc}$	1.516 *ms*
	$T_{child\text{-}proc}$	0.0081 *ms*
	$T_{master\text{-}RPC}(k)$	0.01 *ms*, k = 10 0.011 *ms*, k = 50 0.031 *ms*, k = 1000
	$t_{comparison}$	0.191 μs
	t_{base}	0.28 μs
	$t_{per\text{-}context\text{-}switch}$	2.105 μs
	$ncs_{base}(k)$	56.490, k = 10, 50 97.728, k = 1000
	$ncs_{per\text{-}slave}(k)$	1.917, k = 10, 50 3.316, k = 1000
Network	$ST_{network}(k)$	0.129 *ms*, k = 10 0.222 *ms*, k = 50 0.318 *ms*, k = 1000

Copyright © 2013 Kyu-Young Whang et al.

Measurement of various components of master time

k: the number of records retrieved for a top-*k* query (10, 50, 1000)

ns: the number of slave nodes

$$T_{parent\ -proc} = 1.516\ ms\ ^{\dagger} \qquad T_{child\ -proc} = 0.0081\ ms\ ^{\dagger} \qquad T_{master\ -RPC}(k) = \begin{cases} 0.01\ \text{ms}, & k = 10 \\ 0.011\text{ms}, & k = 50 \\ 0.031\text{ms}, & k = 1000 \end{cases}\ ^{\ddagger}$$

$$T_{merge}(k, ns) = k \times \left(\lceil \log_2(ns) \rceil \times t_{comparisio\ n}\ ^{\dagger\ddagger} + t_{base}\ ^{\ddagger\ddagger} \right)$$

$$t_{comparisio\ n} = 0.191\ \mu s, \qquad t_{base} = 0.28\ \mu s$$

$$T_{context\ -switch}(k, ns) = t_{per\ -context\ -switch} \times \left(ncs_{base}(k)^{\dagger\ddagger\ddagger} + (ns \times ncs_{per\ -slave}(k)^{\ddagger\ddagger\ddagger}) \right)$$

$$t_{per\ -context\ -switch} = 2.105\ \mu s$$

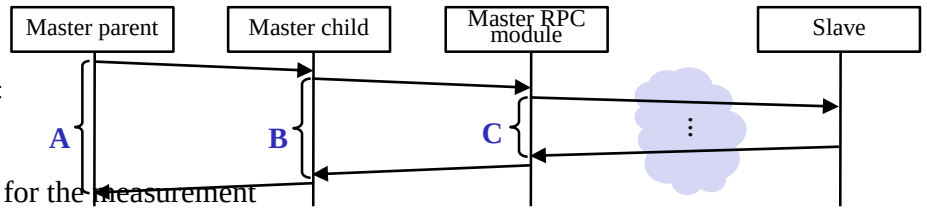
$$ncs_{base}(k) = \begin{cases} 56.490, & k = 10, 50 \\ 97.728, & k = 1000 \end{cases} \qquad ncs_{per\ -slave}(k) = \begin{cases} 1.917, & k = 10, 50 \\ 3.316, & k = 1000 \end{cases}$$

† We measure the total time for a top-10 single keyword query in each module and subtract time consumed by other modules

$T_{parent\text{-}proc}$ = A – B

$T_{child\text{-}proc}$ = B – C

$T_{master\text{-}RPC}$ = master-CPU time of C ‡



‡ See master-CPU time (M) of page 30 for the measurement

†‡ Comparison time

‡‡ Initial cost to merge (read stream, copy the result, etc.)

†‡‡ Initial number of context switches

‡‡‡ Number of context switches per slave

Copyright © 2013 Kyu-Young Whang et al.

- Measurement of the network transfer time of query results

We assume

1. **Network and OS work in parallel**

2. CPU time of OS is the same in the master and in the slave, i.e.,

   Network time = (C – M – S) + 2O

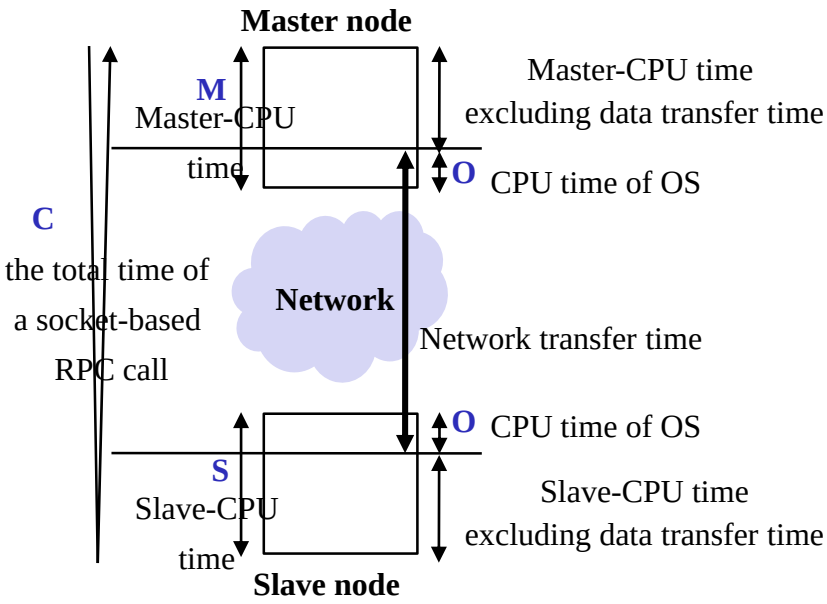
- C is obtained as (end time – start time) of the RPC call at the master.

- M, S are measured at the master, slave, respectively while the program that connects and transfers data to the slave is running on the master.

- M – O is obtained by removing data transfer part from the program.

(M, S, M – O are measured using the 'time' facility, which can measure the total CPU time that a process spent in the kernel mode or user mode.)

Communication Model



Copyright © 2013 Kyu-Young Whang et al.

Service Time of System Component

ST : service time (we assume that the each module's service time for a query is fixed)

ns: the number of slave nodes

α: the proportion of the master CPU time in the master time ($\alpha \leq 1$)

$$ST_{master}(k, ns) = T_{parent\text{-}proc} + (T_{child\text{-}proc} + T_{master\text{-}RPC}(k)) \times ns + T_{merge}(k, ns) + T_{context\text{-}switch}(k, ns) \quad (4)$$

$$ST_{master\_CPU}(k, ns) = ST_{master}(k, ns) \times \alpha \quad (5)$$

$$ST_{master\_memory\_bus}(k, ns) = ST_{master}(k, ns) \times (1 - \alpha) \quad (6)$$

$$ST_{network}(k) = \begin{cases} 0.129\ ms, k = 10 \\ 0.222\ ms, k = 50 \\ 0.318\ ms, k = 1000 \end{cases}^{\dagger} \quad (7)$$

† See the network transfer time in p.45 for the measurement

Copyright © 2013 Kyu-Young Whang et al.

Average Number of Customers in the System

L: average number of customers in the system

λ: the arrival rate of queries

λ': the weighted arrival rate of queries

nm: the number of master nodes

ncm: the number of CPUs per master

ns: the number of slave nodes

nh: the number of network hubs

$$L(\lambda, ST) = \frac{\lambda^2 \, E[ST^2]}{2(1 - \lambda \, E[ST])} + \lambda E[ST]$$

$$L_{master\_CPU}(\lambda, nm, ncm, ns) = L\left(\lambda'_{master\_CPU}(\lambda, nm, ncm), ST_{master\_CPU}(k = 10, ns)\right) \quad (8)$$

$$L_{master\_memory\_bus}(\lambda, nm, ns) = L\left(\lambda'_{master\_memory\_bus}(\lambda, nm), ST_{master\_memory\_bus}(k = 10, ns)\right) \quad (9)$$

$$L_{network}(\lambda, ns, nh) = L\left(\lambda'_{network}(\lambda, ns, nh), ST_{network}(k = 10)\right) \quad (10)$$

Copyright © 2013 Kyu-Young Whang et al.

Expected Total Query Response Time

sct: search condition type (single keyword, multiple keyword, limited search)
k: the number of records retrieved for a top-*k* query (10, 50, 1000)
λ: the arrival rate of queries
nm: the number of master nodes *ncm*: the number of CPUs per master
ns: the number of slave nodes *nh*: the number of network hubs

$$E[X] = \frac{L}{\lambda}$$

X: sojourn time in the system, i.e., the total time a customer spends in the system (waiting time + service time)
λ: the arrival rate
L: average number of customers in the system

$$t_{parallel\ -n-nodes}(sct, k, \lambda, nm, ncm, ns, nh^{\dagger})$$

$$= \max \left(\left(\frac{L_{master\ -CPU}(\lambda, nm, ncm, ns)}{\lambda'_{master\ -CPU}(\lambda, nm, ncm)} + \frac{L_{master\ -memory\ -bus}(\lambda, nm, ns)}{\lambda'_{master\ -memory\ -bus}(\lambda, nm)} \right) \times w_{master}(k), \right.$$
$$\left. \frac{ns}{nh} \times \frac{L_{network}(\lambda, ns, nh)}{\lambda'_{network}(\lambda, ns, nh)} \times w_{network}(k) \right)^{\ddagger}$$
$$+ t_{slave\ -\max\ -time}(sct, k, \lambda, ns) \qquad (12)$$

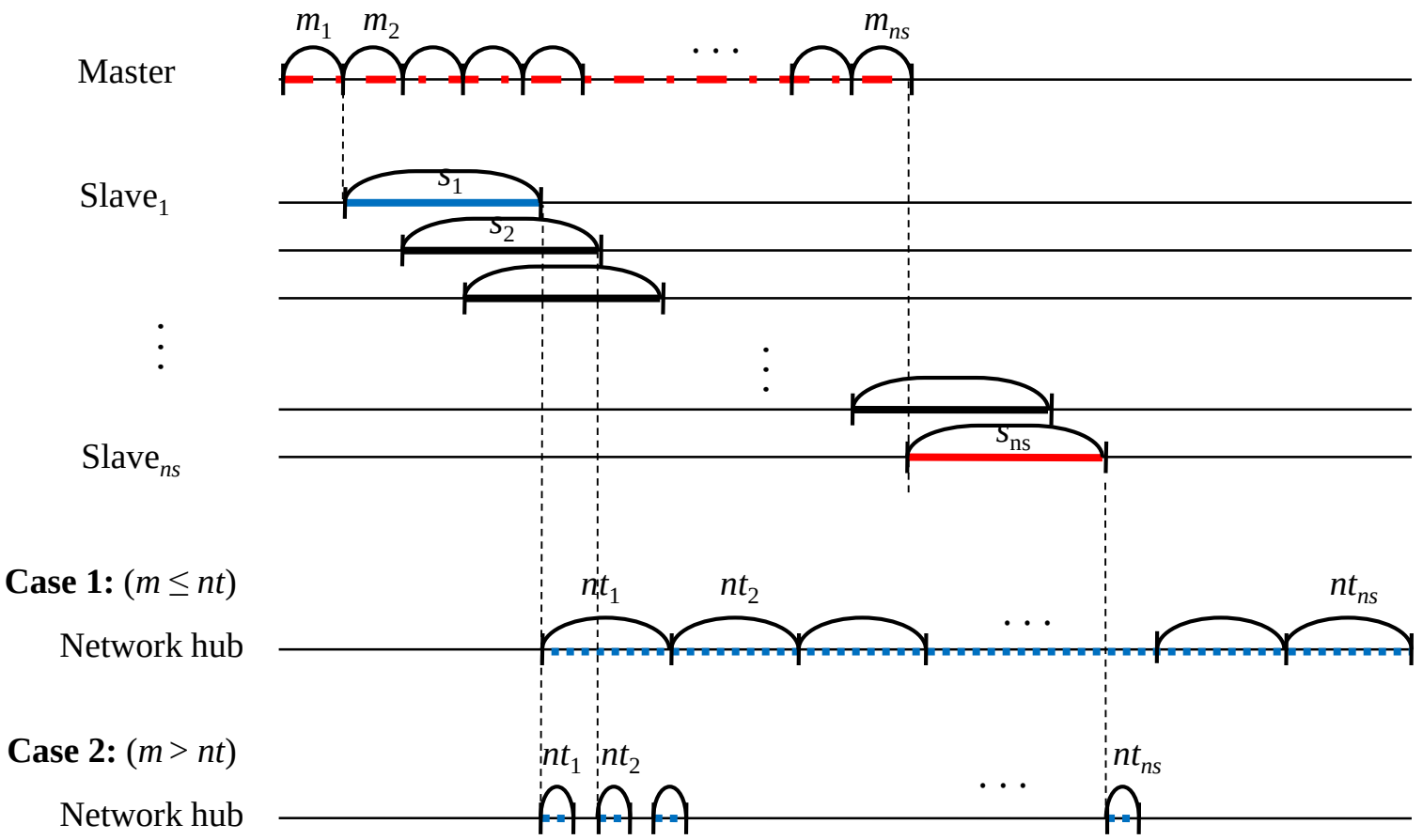
† We assume that a master's I/O bus speed is fast enough to handle multiple LAN cards and each LAN card has a receive buffer

Measure of the Accuracy for the Performance Model

$$estimation\ error = \frac{\left| \begin{matrix} estimated\ average \\ time\ of\ a\ query \end{matrix} - \begin{matrix} measured\ average \\ time\ of\ a\ query \end{matrix} \right|}{\begin{matrix} measured\ average \\ time\ of\ a\ query \end{matrix}}$$

Copyright © 2013 Kyu-Young Whang et al.

‡ Max of master time and network transfer time since the two are processed in parallel



Copyright © 2013 Kyu-Young Whang et al.

Experimenal Model (Slave Max Time)

Algorithm *Partitioning_Method* for estimating the *Slave Max Time*:

Input: (1) Q: the query set,

(2) r: the number of repetitions of the query set execution

(3) np: the number of slaves of the reference system

(4) ns: the number of slaves of the target system

Output: The estimated slave max time for each query in Q

Algorithm:

Step1. Generate a sequence of slave sojourn times for each query:

1.1 Execute Q for r times at semi-cold start by using the np-node system and measure the slave sojourn times.

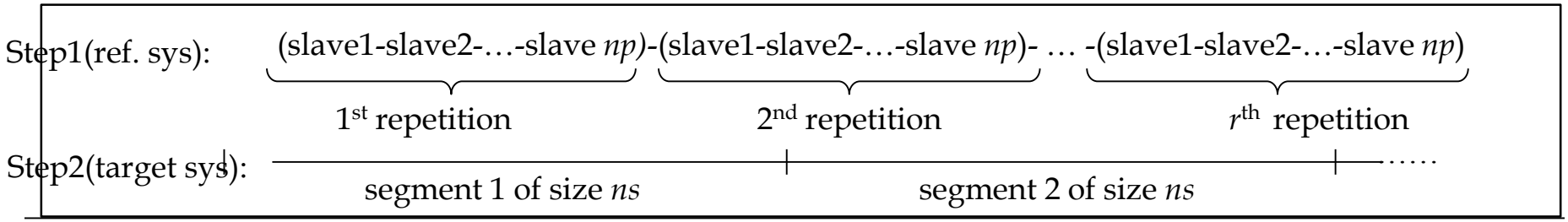
1.2 For the ith query in Q, make a sequence of the slave sojourn times as $< t_{i,1,1}, t_{i,1,2}, \ldots, t_{i,1,m}, t_{i,2,1}, \ldots, t_{i,2,m}, \ldots, t_{i,r,1}, \ldots, t_{i,r,np} >$, where $t_{i,p,q}$ is the slave sojourn time for the ith query in the pth repetition at the qth slave.

Step2. Estimate the average slave max time for ns slaves:

For each sequence obtained in Step1,

2.1 Partition the sequence into segments of size ns.

2.2 Find the maximum value per segment and average those values.



Copyright © 2013 Kyu-Young Whang et al.

Experiments

- Query Generation for Measurement
  - Generating 10,000 random queries at a rate of 1~24 million queries/day/set according to the specified query mix
  - Query generation: Poisson arrival
  - Query sets
    - Single top-10
    - Query-mix

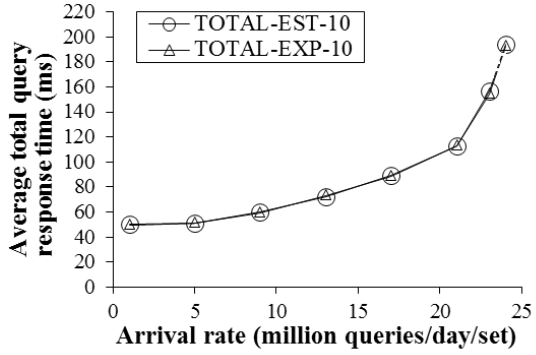
Copyright © 2013 Kyu-Young Whang et al.

- Experimental setting
  - Master
    - one Linux machine (one Quad-Core 3.0GHz CPUs, 6GB RAM)
  - Slaves
    - four Linux machines (two Dual-Core 3.0GHz CPUs, 4GB RAM)
    - one Linux machine (one Quad-Core 2.5GHz CPU, 4GB RAM)
    - five Linux machines (one Quad-Core 2.4GHz CPU, 8GB RAM)
    - four disk arrays (AS-2400~AS-2500, 0.9TB~3.9TB, RAID5, 200MB/s bandwidth, 512MB~1GB cache, average 59.5 MB/s disk transfer rate, 13 disks (arms) + 1 parity disk + 1 hot spare)
    - one disk array (TN-6416S, 13TB, RAID5, 4Gbit/s bandwidth, 512MB cache, average 83.3MB/s disk transfer rate, 13 disks (arms) + 1 parity disk)
    - five internal disk arrays (B110i, 5TB, 768MB/s bandwidth, 81.2MB/s disk transfer rate, 10 disk (arms) + 1 parity disk)
  - Network
    - eleven gigabit LAN cards(Intel 82574L dual-port(1), Intel 82541GI single-port(5), HP NC326i dual-port(5))
    - a gigabit hub (HP 1410-24G, 1000Mbps, 24port)
  - Data
    - 228 million web documents each of 16 KBytes
    - Each slave indexes 22.8 million web documents (Note: A slave is capable of indexing 100 million documents)

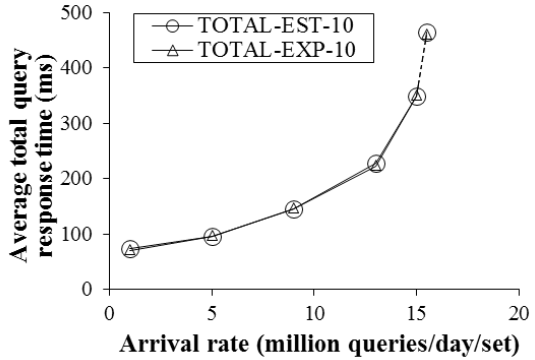
Copyright © 2013 Kyu-Young Whang et al.

Accuracy of the Performance Model

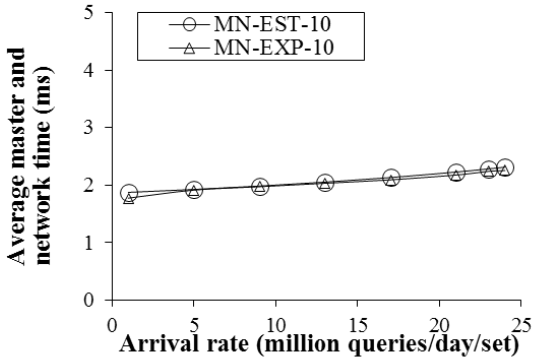
- The estimated and experimental results of the ten-node system (*ns*=10) as the query arrival rate is varied.



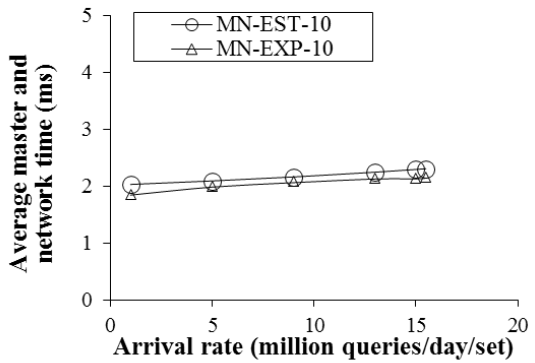
(a) Total, SINGLE-10-ONLY.



(b) Total, QUERY-MIX.



(c) Master and network only, SINGLE-10-ONLY .



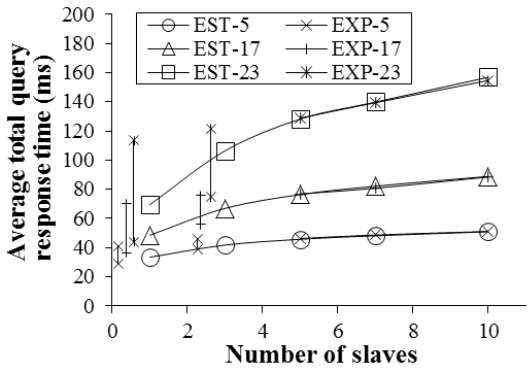
(d) Master and network only, QUERY-MIX .

- Maximum estimation error of the average total query response time
  - SINGLE-10-ONLY: 1.77%
  - QUERY-MIX: 2.13%
- The maximum estimation error of the average master and network time (i.e., the part modeled by the queuing model)
  - SINGLE-10-ONLY: 6.29%
  - QUERY-MIX: 10.15%†

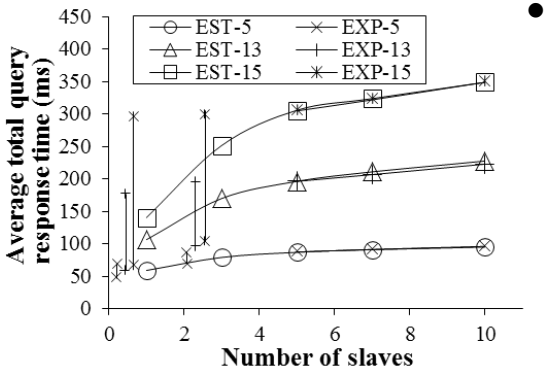
† For sensitivity analysis, we have tested a different query mix having 20% of top-1000 queries obtaining a similar result where the maximum estimation error was 8.64%

Copyright © 2013 Kyu-Young Whang et al.

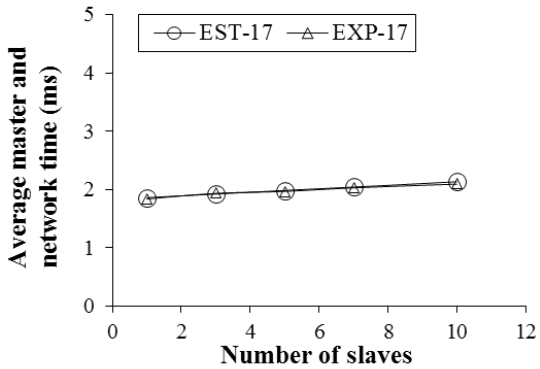
- The estimated and experimental results of the ten-node system (ns=10) as the number of slaves is varied



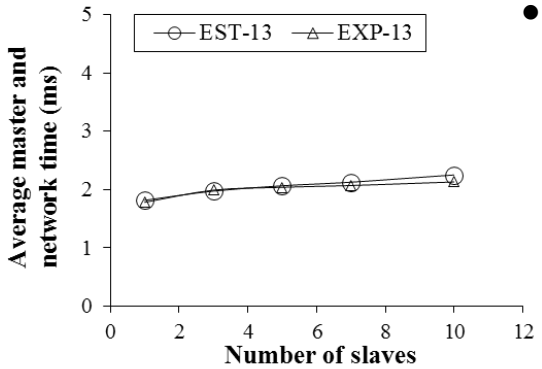
(a) Total, SINGLE-10-ONLY.



(b) Total, QUERY-MIX.



(c) Master and network only, SINGLE-10-ONLY.



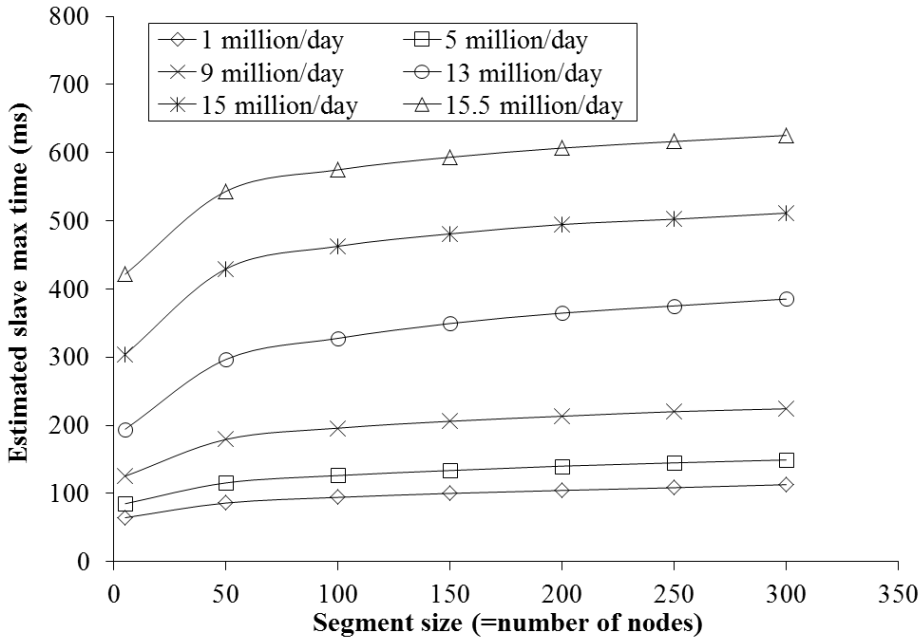
(d) Master and network only, QUERY-MIX.

- Maximum estimation error of the average total query response time is 2.13% when the number of slaves ≥ 5
- Maximum estimation error of the average master and network time is 10.15%

Copyright © 2013 Kyu-Young Whang et al.

Estimation of Slave Max Time

- The estimated slave max time as the segment size is varied (QUERY-MIX, *r*=60, *ns*=5).



- The results show that the expected slave max time increases up to 1.5 ~2 times of the minimum value as the segment size increases
- Interestingly, the slave max time gradually converges to a value less than twice the minimum instead of increasing indefinitely

Copyright © 2013 Kyu-Young Whang et al.

Performance Projection of a Real-World-Scale (300-Node) ODYS

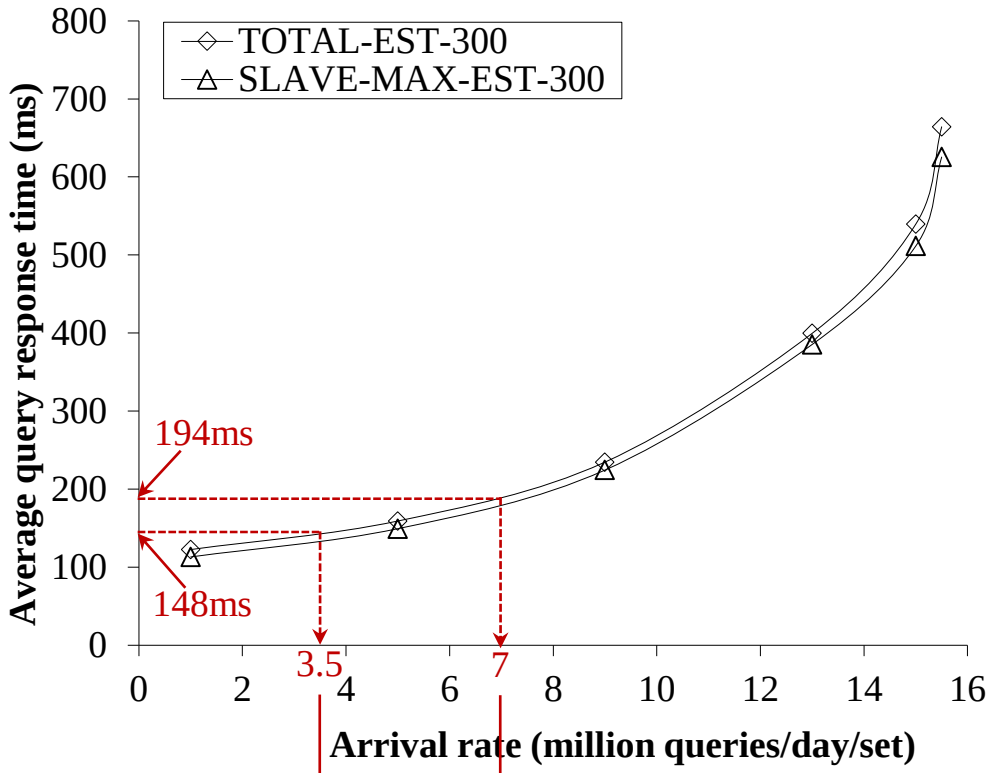
- In the estimation, one ODYS set consists of 4 masters, 300 slaves, and 11 Gbit network hubs
  - Each Master: one quad-core 3.06 GHz CPU
  - Each Slave: two dual-core 3.0 GHz CPU, 4 Gbytes of main memory, and 13×300 Gbytes SATA hard disks

- We select the number of masters (4) and network hubs (11) to make the queue lengths of master memory and network hubs similar to each other to avoid bottlenecks

- The experiments show that our approach is capable of providing a commercial-level service with a rather small number of nodes

Copyright © 2013 Kyu-Young Whang et al.

- The projected average response time of ODYS for real-world-scale service (a 300-node system)



- Query load: 1 billion queries/day
- Web pages indexed: 30 billion (100 million Web pages/slave)
- Nodes required: 43,472 (for 194ms/query)
- Node required: 86,944 (for 148ms/query)

Requires 143 sets of 304 nodes = 43,472 nodes

Requires 286 sets of 304 nodes = 86,944 nodes

Copyright © 2013 Kyu-Young Whang et al.

Performance Comparisons

- Search engines
  - ODYS
  - Search engine A
  - Search engine B
  - Search engine C

- Measures
  - For ODYS, we estimate the query processing time for a 300-node parallel configuration using the performance model
  - For the search engine B, we measure the query processing time that is printed on the result page
  - For the search engines A and C, we measure the query processing time † using the API call since the search time is not printed on the result page

† We subtract the round-trip (ping) time (170ms for the search engine A, and 315ms for the search engine C [Han]) as the network cost.

Copyright © 2013 Kyu-Young Whang et al.

Query Processing Time for Single-Keyword Queries

web search engine / the number of results to retrieve	ODYS 300-node system (λ = 7.0million queries/day)			Search engine A		
	10	50	1,000	10	50	1000
keyword	flower					
the first query (sec)	0.190	0.189	0.207	0.767	0.689	0.861
next 4 queries avg'd (sec)				0.455	0.475	0.596

web search engine / the number of results to retrieve	Search engine B			Search engine C		
	10	50	1,000	10	50	1,000
keyword	flower					
the first query (sec)	0.350	0.600	1.690	0.435	0.935	0.966
next 4 queries avg'd (sec)	0.155	0.255	1.383	0.142	0.601	0.536

Copyright © 2013 Kyu-Young Whang et al.

Conclusions

- We have shown that a massively parallel search engine capable of processing real-world scale data and query loads can be implemented using a DB-IR tightly integrated parallel DBMS— providing higher functionality

- We have presented a detailed implementation (a ten-node system).
  - Masters: ODYS Parallel-IR Master
  - Slaves: Odysseus Object-Relational DBMS equipped with DB-IR tight integration

Copyright © 2013 Kyu-Young Whang et al.

- We have proposed a performance model and validated it through extensive experiments
  - Hybrid model
    - Analytic: estimate the master and network time by using a queuing model
    - Experimental: estimate the slave max time through actual measurement with a small-scale reference system
    - The hybrid model is helpful in realistically estimating the performance of a system by using limited resources without actually building a large-scale system
  - Model validation
    - The estimation by the model with a five-node reference system vs. the results measured by the one-to-ten node system
    - The estimation error of the total query response time of the ten-node system is less than 2.13%
    - We argue that the model is accurate since the bulk of the total query response time is spent at the slave and we derive the slave max time by measurement from the reference system

Copyright © 2013 Kyu-Young Whang et al.

- We have estimated the performance of ODYS for real-world-scale data and query loads
  - ODYS is capable of handling 1 billion queries/day for 30 billion Web pages with an average query response time of
    - 194 ms with 43,472 nodes
    - 148 ms with 86,944 nodes
  - This result clearly demonstrates the scalability and efficiency of the proposed architecture
  - The result is even more marked since these are conservative results from a semi-cold start reflecting a lower-bound performance (The warm-start performance is approximately five times faster than semi-cold start performance)

Copyright © 2013 Kyu-Young Whang et al.

References

- **[Abi05]** Abiteboul, S. et al., "The Lowell Database Research Self-Assessment," *Comm. of ACM*, Vol. 48, No. 5, pp. 111-118, May 2005.
- **[Abo09]** Abouzeid, A. et al., "HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads," In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pp. 922-933, Aug. 2009.
- **[Cha06]** Chang F., et al., "BigTable: A Distributed Storage System for Structured Data," In *Proc. 6th Symposium on Operating Systems Design and Implementation* (*OSDI '06)*, pp. 205-218, Dec. 2006.
- **[Cha05]** Chaudhuri, S., Ramakrishnan, R., and Weikum, G., "Integrating DB and IR Technologies: What is the Sound of One Hand Clapping?," In *Proc. 2nd Biennial Conf. on Innovative Data Systems Research*, Asilomar, California, pp. 1-12, Jan. 2005.
- **[Coh09]** Cohen J., Dolan B., and Dunlap M., Joseph Hellerstein, and Caleb Welton, "MAD Skills: New Analysis Practices for Big Data," In *Proc. 35th Int'l Conf. on Very Large Data Bases (VLDB)*, pp. 1481-1492, Aug. 2009.
- **[Coo08]** Cooper B. et al., "PNUTS: Yahoo!'s Hosted Data Serving Platform ," In *Proc. 34th Int'l Conf. on Very Large Data Bases(VLDB)*, pp. 1277-1288, Aug. 2008.
- **[Dea04]** Dean, J., and Ghemawat, S., "MapReduce: Simplified Data Processing on Large Clusters," In Proc. Symposium on Operating Systems Design and Implementation (OSDI), pp. 137-150 , Dec. 2004.
- **[Dea09]** Dean, J., "Challenges in Building Large-Scale Information Retrieval Systems," In *Proc. ACM Int'l Conf. on Web Search and Data Mining (WSDM)* (an invited talk), p. 1, Feb. 2009.
- **[Fur08]** Furman J., et al., "Megastore: A Scalable Data System for User Facing Applications," *In 2008 ACM SIGMOD Int'l Conf. on Management of Data*, June 2008.

Copyright © 2013 Kyu-Young Whang et al.

- **[Ghe03]** Ghemawat, S., Gobioff, H., and Leung, S., "The Google file system," In *Proc. 19th ACM Symposium on Operating Systems Principles*, pp. 29-43, Oct. 2003.
- **[Guo03]** Guo, L. et al., "XRANK: Ranked Keyword Search over XML Documents," In *Proc. 2003 ACM SIGMOD Int'l Conf. on Management of Data*, San Diego, California, pp. 16-27, June 2003.
- **[Had]** Hadoop, http://hadoop.apache.org
- **[Hal03]** Halverson, A. et al., "Mixed Mode XML Query Processing," In *Proc. 29th Int'l Conf. on Very Large Data Bases*, Berlin, Germany, pp. 225-236, Sept. 2003.
- **[Kun13]** Kunder, M., http://www.worldwidewebsize.com.
- **[Len04]** Lentz, A., "MySQL Storage Engine Architecture," In MySQL Developer Articles, MySQL AB, May 2004.
- **[Nie10]** Nielsenwire, "Nielsen Reports February 2010 U.S. Search Rankings," Nielsen Report, Mar. 15, 2010 (available at http://blog.nielsen.com/nielsenwire/online_mobile/nielsenreports-february-2010-u-s-search-rankings).
- **[Ols08]** Olston C., Reed B., Srivastava U., Kumar R., and Tomkins A., "Pig Latin: A Not-So-Foreign Language for Data Processing," In *Proc. 2008 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 1099–1110, June 2008.
- **[Ric99]** Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto, *Modern Information Retrieval*, ACM Press, Addison-Wesley, 1999.
- **[Sto10]** Stonebraker, M. et al., "MapReduce and Parallel DBMSs: Friends or Foes?," *Communications of the ACM (CACM)*, pp. 64-71, Jan. 2010.

 Copyright © 2013 Kyu-Young Whang et al.

- **[Thu09]** Thusoo A. et al., "Hive - A Warehousing Solution Over a Map-Reduce Framework," In *Proc. 35th Int'l Conf. on Very Large Data Bases (VLDB)*, pp. 1626-1629, Aug. 2009.
- **[Ver]** Vertica, http://www.vertica.com.
- **[Wei07]** Weikum, G., "DB&IR: both sides now," In Proc. *2007 ACM SIGMOD Int'l Conf. on Management of Data*, pp. 25-30, Beijing, China, June 12-14, 2007.
- **[Wha02]** Whang, K. et al., An Inverted Index Storage Structure Using Subindexes and Large Objects for Tight Coupling of Information Retrieval with Database Management Systems, U.S. Patent No. 6,349,308, Feb. 19, 2002, Application No. 09/250,487, Feb. 15, 1999.
- **[Wha03]** Whang, K., "Tight Coupling: A Way of Building High-Performance Application Specific Engines," a presentation at the panel Next-Generation Web Technology and Database Issues, *the 8th International Conference on Database Systems for Advanced Applications (DASFAA 2003)*, Kyoto, Japan, URL:http://db-www.aist-nara.ac.jp/dasfaa2003/ppt.html, Mar. 2003.
- **[Wha05]** Whang, K., et al., "Odysseus: a High-Performance ORDBMS Tightly-Coupled with IR Features," In *Proc. 21st Int'l Conf. on Data Engineering*, Tokyo, Japan, pp. 1104-1105, April 2005. **This paper received the Best Demonstration Award.**
- **[Wha11]** Kyu-Young Whang, "Large-scale Data Management: NoSQL vs. Parallel DBMS," Panel on Challenges in Managing and Mining Large, Heterogeneous Data, *DASFAA*, http://www.cintec.cuhk.edu.hk/DASFAA2011/doc/KyuYoungWhang-Panel.pdf, Apr. 24, 2011.
- **[Wha13]** Whang, K. et al., "DB-IR Integration Using Tight-Coupling in the Odysseus DBMS," submitted for publication, 2013.

 Copyright © 2013 Kyu-Young Whang et al.