Experiments on Fragmentation

Hao, Wei

December 18, 2017

1 E2017120401: Baseline Performance Evaluation

1.1 Purpose

To understand the baseline performance by using the queries in ADBIS submission for XMark600.

1.2 Experiment Design

We run a BaseX server *Server* for processing XPath queries on specified databases. Server starts by the following command on HaoDesk.

```
java -Xmx4g -xms2g -cp BaseX897.jar org.basex.BaseXServer
```

Note the databases in Server are NOT in main memory mode.

We run a java program *Japp* (that is the class `basex.ORIG` in repository/src/fragmentation/java/basex) on HaoDesk in charge of sending an input query to Server via local network and saving results returned from Server. An input query Query that will be processed in Japp is first rewritten into the following XQuery expression:

```
for $node in db:open('xmark600')Query return $node
```

The results of the input query are stored either in memory or on disk depending on Japp's settings. When in memory, the results will be then discarded after the experiments, while on disk, the results will be preserved in files. One more thing, the maximum available memory for Japp was set to 12 GB.

1.3 Settings

- **Hardware** HaoDesk (see A.1.1)

- **Software** BaseX 6.8.7, Java 1.8.0_151(x64).

- **XML Dataset** XMark600.xml (see Table 5), from which a BaseX databases 'xmark600' is created in a BaseX server using command:
  `create db xmark600 xmark600.xml`

- **Queries** xm1.org – xm6.org (see Table 6).

1.4 Experiment Results

Timing The execution time is measured in Japp. The time period between starting sending a query and finishing receiving the results is measured as execution time. Each query is evaluated 5 times.

Process Results We disgard the execution time of the first run and take the average of the rest as the final execution time listed in Table 1.

Original Experiment Data

Table 1: Experiment Results of E2017120401.

query	storage	time(s)	nodes (M)	result size (MiB)
xm1.org	disk	3122.88	55	57,267
	memory	N/A		
xm2.org	disk	0.01	0	0
	memory	0.01		
xm3.org	disk	63.32	6.5	880
	memory	63.42		
xm4.org	disk	101.51	0.6	1,511
	memory	101.01		
xm5.org	disk	74.85	35.9	944
	memory	75.05		
xm6.org	disk	71.29	1.3	1,289
	memory	70.07		

Table 2: My caption

Key	Prefix (s)	Suffix (s)					Merge (s)	Original (s)	Results Size	
		P=1	P=2	P=3	P=4	P=8			hit nodes(M)	in MiB
xm1.dps								3122	5.5	57267
xm2.dps								0.01	0	0
xm3.dps	0.3	113	58.5	30.1	20.2	13.5	6	63.3	6.5	880
xm4.dps								101	0.6	1511
xm5.dps	0.3	243	67.5	48.9	37.8	31.8	10	74.9	35.9	944
xm6.dps								70	1.3	1289

All the original results containing execution time of queries and scripts used in the experiments are stored to `experiments/E2017120401` relative to the current folder that stores this report.

Note: The result of xm2.org is always empty. This is because there is no `incategory` node with attribute that has the content of `category52`.

1.5 Observations

- **Storage has small influence on execution time**

  We noticed one thing that the execution time is pretty similar for all the available queries. This is because the bottleneck is on the worker's side but not on the master's side. For example, for xm4.org, it takes 113s to receive about 1500 MB data, i.e. around 13.36 MB/s, which is much slower than the maximum speed of both memory and disk. Thus, the performance are much similar. We also notice that for some queries such as xm3.org and xm5.org, in-memory case is even a bit slower than on-disk case, one possible explanation is that the time was taken by calling System.gc().

- **The execution time is steady**

  Compared with the ADBIS study, the execution time of each run is more steady. Our explanation to this result is that for very large scale of data, the fluctuation has a weaker influence on the execution time (increased from milliseconds to seconds).

2 E2017120501: Data Partitioning on Fragmented XML Data

2.1 Settings

The settings of this experiments are listed below.

Hardware There were 1 master (HaoDesk) and 16 workers (matsu-lab00 – matsu-lab15) used in the experiments listed in C.2.

Software The input XML data set was xmark600(see C.2), which was fragmented into 16 fragments with maxisze=4M. On each worker, there ran a BaseX server (Verion: 8.6.7). For each fragment, a BaseX database instance, namely ‘xmark600_16_4Mẃas created in main memory on each worker. For the data partition, the number of partions(P) were 1, 2, 3, 4, 8. The intermediate database for holding the results of prefix query were named ‘xmark600_16_4M_tmp’.

XQuery Expressions Two XPath expressions XQ1 and XQ2 for prefix and suffix queries are listed in Section C.2.

2.2 Confirming Correctness

The number of hit nodes, the order of hit nodes and results size of the final output of queries have been checked and confirmed to be correct. All the successfully evaluated queries have the same number of hit nodes in original order. Some may be different in size, such as xm3a.dps. Its original result size was 922,270,281, but in this experiment, it is 883,253,777. This dramatical difference is caused by line-break. The previous experiments were done on Windows, while the current on Linux. Since BaseX using ‘\r\n’ on Windows, while ‘\n’ on Linux for line-break, there is one byte difference for each new line. We also found that the query has 6,502,751 hit nodes and there are six lines in every hit node, such as:

```
<bidder>
  <date>08/04/1999</date>

    <time>11:15:36</time>
  <personref person="person17793"/>
```

Table 3: Execution time for xm3.dps

query	P=1	P=2	P=3	P=4	P=6	P=8	P=12
prefix	$\approx$ 0.3s						
suffix	113.2s	58.5s	30.1s	20.2s	15.8s	13.5s	11.3s
merge	$\approx$ 6s						

```
  <increase>7.50</increase>
</bidder>
```

We then have 883,253,777 + 6,502,751 * 6 - 922,270,281 = 2. These two bytes are extra '\n'. So the sizes are the same.

2.3 Discussion on Queries

xm1.dps Since the results of xm1.dps are larger than the memory of HaoDesk. This query was not evaluated. This is a design choice about how to process the results. Based the previous evaluation, one possible way is to stored the unordered intermediate results in files and then concatenate these files.

xm2.dps Not tested for two reasons. Firstly, the prefix query of xm2.dps (as well as xm4.dps) cannot be processed by the current XQuery expression XQ1 (will be modifed and tested soon). Secondly, there is no results as shown in experiment E20170401. One proposal is to make a minimal change to the query, such as change `category52` to `category324329`, which exists in XMark600.

xm3.dps The results of xm3.dps are shown in Table 2.3. (xm3.org = 63.32s)

xm4.dps Not tested for the same reason as the first one of xm2.dps.

xm5.dps The query xm5a.dps was the targe query in the previous design. However, due to the insufficient number of nodes in the results of prefix query to be allocated to 16 computers[1] , I changed it to xm5b.dps (xm5c.dps was also evaluated, but it took 1480s (P=4), which is too long and thus ignored). The results of xm5b.dps is listed in Table 2.3 (xm5.dps = 75.05s).

xm6.dps Failed to evaluate them with data partitioning. The reason is the same as xm5a.dps (both `/site/regions` and `/site/regions/*[...]/item` were tested).

[1] Detailed explanation: An error message "database 'xmark600_16_4M_tmp' has no node with pre value 5." was encountered when executing a suffix query on an intermediate database. The database had the content of `<root><part> ...</part></root>` , where there is only one `part` node. In a suffix query when P = 2, it is then to process "pre value 5" in XQ2, which refers to the second `part` node. However, since there is no second `part` node, the error occurred.

Table 4: Execution time for xm5b.dps

query	P=1	P=2	P=3	P=4	P=6	P=8	P=12
prefix	≈ 0.3s						
suffix	243s	67.5s	48.9s	37.8s	33.4s	31.8s	28.2s
merge	≈ 10s						

2.4 Efficiency of Parsing Intermediate Results of Suffix Query

A important method in the implemenation that affects the whole performance is `basex.PreValueReceiver.process(InputStream input)` used to parse the received results of suffix query, (i.e. PRE value + content). It takes the results of suffix query and returns an QueryResultsPre instance with a list of PRE values and a list of string content (of the same size). A simple experiment were done to evaluate the parsing speed. In the experiment, it took 465 ms to parse 52,757 KiB data with 704,430 nodes, i.e. it can process 100 MiB data per second, which basically reach the maximum network speed and thus should be sufficient for not being a bottleneck.

A Environments

A.1 Computers

A.1.1 HaoDesk

CPU: Intel Core (TM) i7 3930K@3.2 GHz (turbo to 3.8 GHz), 6 cores 12 threads
Memory: 32 GB DDR3 1333 GHz
Disk: 256GB SSD + 4TB HDD
Windows 7 Professional SP1 64bit

A.1.2 HaoHome

CPU: Intel Core (TM) i7 3930K@3.2 GHz (turbo to 3.8 GHz), 6 cores 12 threads
Memory: 16 GB DDR3 1333 GHz
Disk: 256GB SSD + 2TB HDD
Windows 10 64bit

B DataSets

B.1 XMark Dataset

The XMark datasets are listed in Table 5.

Table 5: Statistics of XMark datasets

key	# elements	# attributes	# context	total nodes	file size
xmark1	1,666,315	381,878	1,173,732	3,221,925	113.06 MB
xmark600	1,002,327,042	229,871,111	705,824,967	1,938,023,120	66.99 GB

Table 6: Original XPath queries on XMark datasets.

key	query
xm1.org	`/site//*[name(.)="emailaddress" or name(.)="annotation" or name(.)="description"]`
xm2.org	`/site//incategory[./@category="category52"]/parent::item/@id`
xm3.org	`/site//open\_auction/bidder[last()]`
xm4.org	`/site/regions/*/item[./location="United States" and ./quantity > and ./payment="Creditcard" and ./description and ./name]`
xm5.org	`/site/open\_auctions/open\_auction/bidder/increase`
xm6.org	`/site/regions/*[name(.)="africa" or name(.)="asia"]/item/description/parlist/listitem`

C Queries

C.1 XPath Quereis

The original XPath queries for XMark datasets are listed in Table 6. The data partitioning XPath queries of Table 6 is listed in Table 7.

C.2 XQuery Expressions

The two XQuery expressions are used for processing prefix query and suffix query for data partitioning strategy shown in Table C.2 and Table C.2 respectively. Note that `{eval_prefix}` will be first replaced by `{prefix}` or `db:open(’{db}’){prefix}` depending on whether the query is optimized.

Table 7: Data partitioning XPath Queries on XMark datasets.

key	query
xm1.dps	pre = /site//*, suf = [name(.)="emailaddress" or name(.)="annotation" or name(.)="description"]

xm2.dps	pre = db:attribute("{db}", "category52"),
	suf = /parent::incategory/parent::item/@id

xm3.dps	pre = /site/open_auctions/open_auction,
	suf = /bidder[last()]

xm4.dps	pre = db:text("{db}", "Creditcard")
	suf = /parent::*:item[parent::*/parent::*:regions /parent::*:site/parent::document-node()] [(*:location = "United States")][0.0 < *:quantity] [*:description][*:name]

xm5a.dps	pre = /site/open_auctions,
	suf = /open_auction/bidder/increase

xm5b.dps	pre = /site/open_auctions/open_auction,
	suf = /bidder/increase

xm5c.dps	pre = /site/open_auctions/open_auction/bidder,
	suf = /increase

xm6.dps	pre = /site/regions,
	suf = /self::*[name(.)="africa" or name(.)="asia"]/item /description/parlist/listitem

Table 8: XQuery Expression for XQ1 prefix part

```
// XQ12: for prefix query
let $d := array { {eval_prefix} !
          db:node-pre(.) } return
for $i in 0 to {P} − 1 return
let $q := array:size($d) idiv {P} return
let $r := array:size($d) mod {P} return
let $part_length := if ($i < $r) then $q + 1
else $q return
let $part_begin  := if ($i <= $r) then ($q + 1) ∗ $i
else $q ∗ $i + $r return
insert node element part {
array:subarray($d, $part_begin + 1, $part_length)
} as last into db:open('{tmpdb}')/root
```

Table 9: XQuery Expression XQ2 for suffix part

```
// XQ2: for suffix query
declare option output:method '$mode';
declare option output:item-separator '[';

for $pre in db:open('{tmpdb}')/root return
let $node := db:open('{db}'){suffix}
return (db:node-pre($node), $node)

let $part_pre := {p}*2 + 1 return
for $pre in ft:tokenize(db:open-pre('{tmpdb}', $part_pre)) return
for $node in db:open-pre('{db}', xs:integer($pre)){suffix})
return (db:node-pre($node), $node)
```