12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16)

Errata Slip

In the paper “Gemini: A Computation-Centric Distributed Graph Processing System” by Xiaowei Zhu, Wenguang Chen, and Weimin Zheng, *Tsinghua University*; Xiaosong Ma, *Hamad Bin Khalifa University* (Thursday session, “Graph Processing and Machine Learning,” pp. 301-316 of the Proceedings), the following changes were made:

1. Table 1 (p. 302) contained incorrect numbers in the second (1-core; OST) column:

Cores	1	24 × 1		24 × 8	
System	OST	Ligra	Galois	PowerG.	PowerL.
Runtime (s)	99.9	21.9	19.3	40.3	26.9
Instructions	525G	496G	482G	7.15T	6.06T
Mem. Ref.	15.8G	32.3G	23.4G	95.8G	87.2G
Comm. (GB)	-	-	-	115	38.1
IPC	1.71	0.408	0.414	0.500	0.655
LLC Miss	8.77%	43.9%	49.7%	71.0%	54.9%
CPU Util.	100%	91.7%	96.8%	65.5%	68.4%

Table 1 original

Cores	1	24 × 1		24 × 8	
System	OST	Ligra	Galois	PowerG.	PowerL.
Runtime (s)	79.1	21.9	19.3	40.3	26.9
Instructions	525G	496G	482G	7.15T	6.06T
Mem. Ref.	8.58G	32.3G	23.4G	95.8G	87.2G
Comm. (GB)	-	-	-	115	38.1
IPC	2.16	0.408	0.414	0.500	0.655
LLC Miss	14.8%	43.9%	49.7%	71.0%	54.9%
CPU Util.	100%	91.7%	96.8%	65.5%	68.4%

Table 1 corrected

Related errors in Section 7.2 (p. 311):

Figure 9:

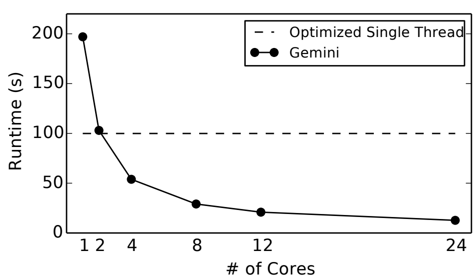


Figure 9 original

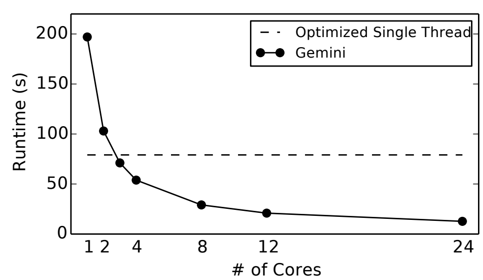


Figure 9 corrected

In the second paragraph:

Original text

“…, Gemini’s number is 3, which is lower than those of other systems measured [33], though Gemini’s 2-core execution time is only 3.1% higher than the optimized single-thread implementation. Considering Gemini’s distributed nature, a COST close to 2 illustrates its optimized computation efficiency and lightweight distributed execution overhead.”

Corrected text

“…, Gemini’s number is 3 (with its 2-core execution time 30.2% higher than the optimized single-thread implementation), which is lower than those of other systems measured [33]. Considering Gemini’s distributed nature, the COST illustrates its optimized computation efficiency and lightweight distributed execution overhead.”

Continues on next page

2. The pseudo-codes in Figures 2-3 (p. 304) contained some errors, with red boxes marking the changes:

```
class Graph<E> {
  VertexID vertices;
  EdgeID edges;
  VertexID [] outDegree;
  VertexID [] inDegree;
  def allocVertexArray<V>() -> V [];
  def allocVertexSet() -> VertexSet;
  def processVertices<A> (
    work: (VertexID) -> A,
    active: VertexSet,
    reduce: (A, A) -> A,
  ) -> A;
  def processEdges<A, M> (
    sparseSignal: (VertexID) -> void,
    sparseSlot: (VertexID, M, OutEdgeIterator<E>) -> A,
    denseSignal: (VertexID, InEdgeIterator<E>) -> void,
    denseSlot: (VertexID, M) -> A,
    reduce: (A, A) -> A,
    active: VertexSet
  ) -> A;
  def emit<M> (recipient: VertexID, message: M) -> void;
};
```

Figure 2 original

```
class Graph<E> {
  VertexID vertices;
  EdgeID edges;
  VertexID [] outDegree;
  VertexID [] inDegree;
  def allocVertexArray<V>() -> V [];
  def allocVertexSet() -> VertexSet;
  def processVertices<A> (
    work: (VertexID) -> A,
    active: VertexSet,
    reduce: (A, A) -> A,
  ) -> A;
  def processEdges<A, M> (
    sparseSignal: (VertexID) -> void,
    sparseSlot: (VertexID, M, OutEdgeIterator<E>) -> A,
    denseSignal: (VertexID, InEdgeIterator<E>) -> void,
    denseSlot: (VertexID, M) -> A,
    active: VertexSet,
    reduce: (A, A) -> A
  ) -> A;
  def emit<M> (recipient: VertexID, message: M) -> void;
};
```

Figure 2 corrected

```
Graph<empty> g (...); // load a graph from the file system
VertexSet activeCurr = g.allocVertexSet();
VertexSet activeNext = g.allocVertexSet();
activeCurr.fill(); // add all vertices to the set
VertexID [] label = g.allocVertexArray <VertexID> ();
def add (VertexID a, VertexID b) : VertexID {
  return a + b;
}
def initialize (VertexID v) : VertexID {
  label[v] = v;
  return 1;
}
VertexID activated = g.processVertices <VertexID> (
  initialize,
  activeCurr
);
```

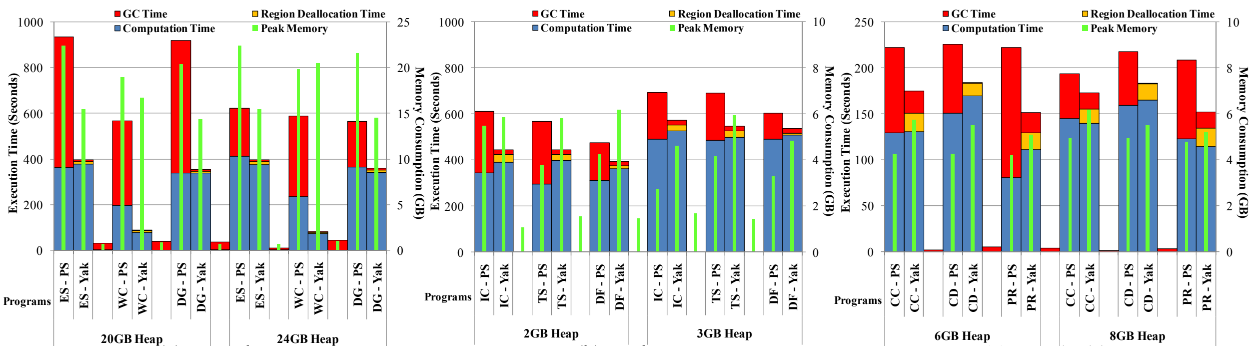
Figure 3 original

```
Graph<empty> g (...); // load a graph from the file system
VertexSet activeCurr = g.allocVertexSet();
VertexSet activeNext = g.allocVertexSet();
activeCurr.fill(); // add all vertices to the set
VertexID [] label = g.allocVertexArray <VertexID> ();
def add (VertexID a, VertexID b) : VertexID {
  return a + b;
}
def initialize (VertexID v) : VertexID {
  label[v] = v;
  return 1;
}
VertexID activated = g.processVertices <VertexID> (
  initialize,
  activeCurr,
  add
);
```

Figure 3 corrected

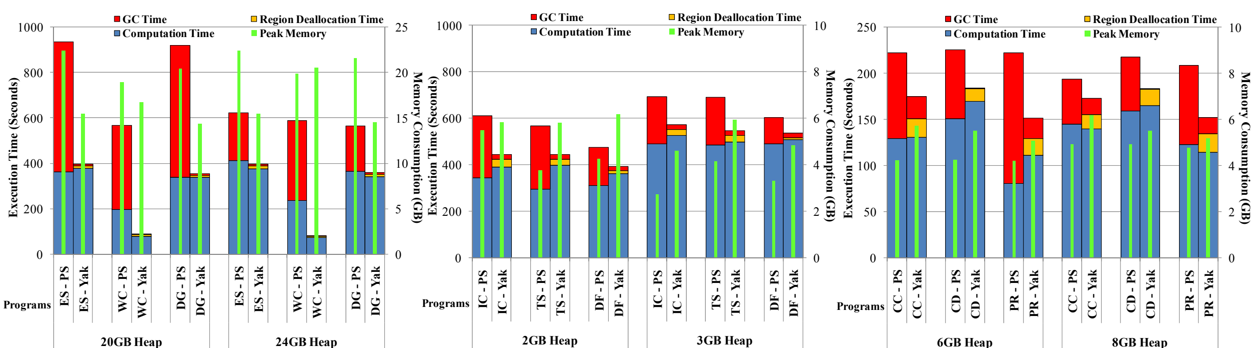
In the paper "Yak: A High-Performance Big-Data-Friendly Garbage Collector" by Khanh Nguyen, Lu Fang, Guoqing Xu, and Brian Demsky; *University of California, Irvine;* Shan Lu, *University of Chicago;* Sanazsadat Alamian, *University of California, Irvine;* Onur Mutlu, *ETH Zurich* (Thursday session, "Languages and Software Engineering," pp. 349-365 of the Proceedings), the following correction was made to Figure 10 (p. 360):

Original:



(a) Hyracks (b) Hadoop (c) GraphChi

Corrected:



(a) Hyracks (b) Hadoop (c) GraphChi

For the paper “Consolidating Concurrency Control and Consensus for Commits under Conflicts by Shuai Mu and Lamont Nelson, *New York University;* Wyatt Lloyd, *University of Southern California;* Jinyang Li, *New York University* (Thursday session, “Fault Tolerance and Consensus,” pp. 517–532 of the Proceedings):

The most up to date and preferred version of this paper is available at http://mpaxos.com/pub/janus-osdi16.pdf. It contains corrections for minor typographic errors as well as changes in the prose and pseudocode for clarity. The notable changes are itemized below:

Removed an unnecessary paragraph break in the Accept phase portion of section 3.2.

A formula in section 3.3 was updated to indicate when a recovery coordinator is guaranteed to observe conflicting transactions dependencies. The formula $(\mathcal{F} \cap \mathcal{M}) \cap (\mathcal{F} \cap \mathcal{M}) \neq \emptyset$ was changed to $(\mathcal{F} \cap \mathcal{M}) \cap (\mathcal{F}' \cap \mathcal{M}') \neq \emptyset$. Extra prime symbols are added to clarify that they are not the same set.

Edited all psuedocode for clarity:

1. The conditions referencing reaching the ‘committing’ status as ‘committing’ were changed to ‘is committing’.

2. The Accept phase of Algorithm 1 is more concise; a reference to parallel message delivery was omitted.

3. Commented pseudocode in Algorithm 2 was removed.

4. The visual format of the psuedocode was adjusted to remove extra spacing.

5. Ballot number is better viewed as state associated with a dependency instead of state associated with status. Therefore, it is extracted from the status as a separate eld.

6. Use ‘ ‘ instead of ‘=’ for assigning.