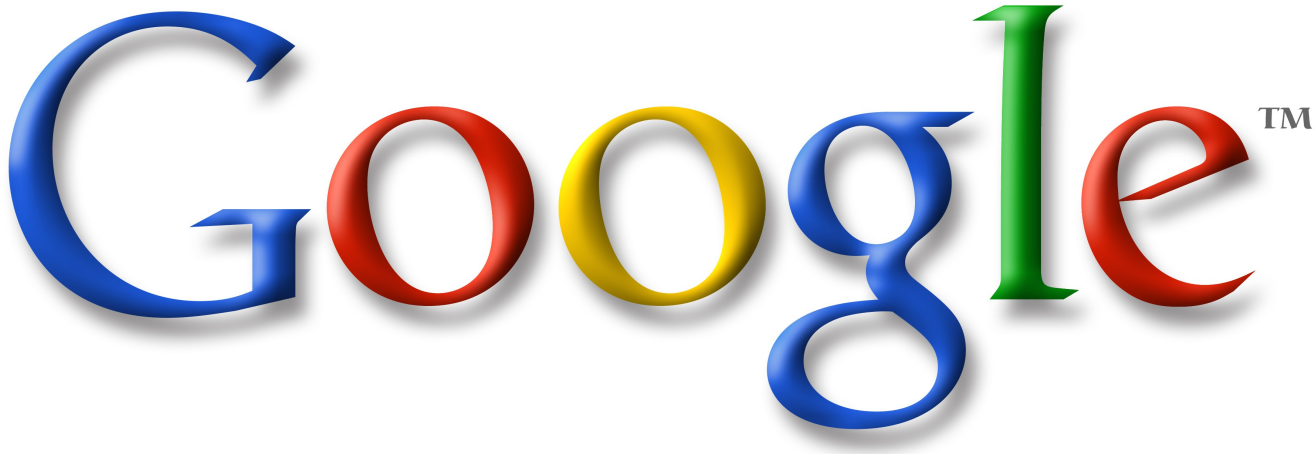
MapReduce a distribuované výpočty



25. listopadu 2008

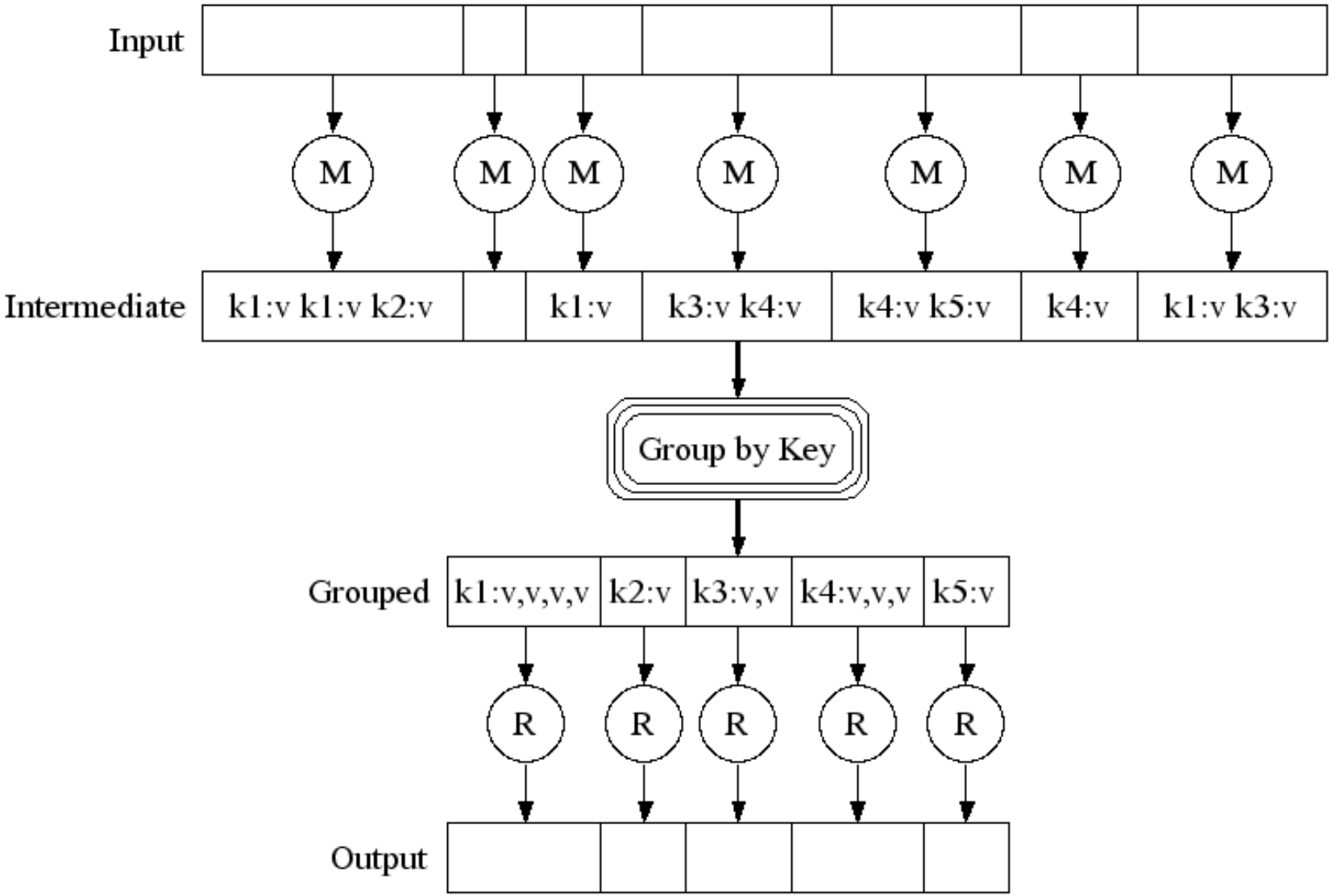
Prostředí

Máme k dispozici následující počítače:

1. dvoujaderné x86 procesory,
2. běžné IDE disky s distribuovaným souborovým systémem,
3. běžný síťový hardware – 100Mb/s nebo 1Gb/s.

Cluster obsahuje tisíce takových počítačů.

Programovací model



Programovací model

Map

Z jedné dvojice `(klíč, hodnota)` vytvoří seznam dvojic `(klíč, dočasná hodnota)`.

```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

Reduce

Z klíče a seznamu dočasných hodnot vytvoří výsledný seznam hodnot tohoto klíče.

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

Programovací model

Map

Z jedné dvojice `(klíč, hodnota)` vytvoří seznam dvojic `(klíč, dočasná hodnota)`.

```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

Reduce

Z klíče a seznamu dočasných hodnot vytvoří výsledný seznam hodnot tohoto klíče.

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

Příklad: počítání výskytů slov

Map

```
map (String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

Reduce

```
reduce (String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Příklad: počítání výskytů slov

Map

```
map (String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");
```

Reduce

```
reduce (String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Další příklady

Počet přístupů na dané URL

Stejně jako počítání výskytů slov.

Výběr vyhovujících položek

Funkce `map` vrátí pouze vyhovující položky, funkce `reduce` nemění seznam hodnot.

Matice webu

```
map (String url, String content):
  for each referenced h in content:
    EmitIntermediate(href, url);

reduce (String url, Iterator sources):
  Emit(sources);
```

Další příklady

Počet přístupů na dané URL

Stejně jako počítání výskytů slov.

Výběr vyhovujících položek

Funkce `map` vrátí pouze vyhovující položky, funkce `reduce` nemění seznam hodnot.

Matice webu

```
map (String url, String content):
  for each referenced h in content:
    EmitIntermediate(href, url);

reduce (String url, Iterator sources):
  Emit(sources);
```

Další příklady

Počet přístupů na dané URL

Stejně jako počítání výskytů slov.

Výběr vyhovujících položek

Funkce `map` vrátí pouze vyhovující položky, funkce `reduce` nemění seznam hodnot.

Matice webu

```
map (String url, String content):
  for each referenced h in content:
    EmitIntermediate(href, url);

reduce (String url, Iterator sources):
  Emit(sources);
```

Další příklady

Invertovaný index

```
map (String docid, String content):
  for each word w in content:
    EmitIntermediate(word, docid);

reduce (String word, Iterator docids):
  Emit(sort(docids));
```

Třídění dat

```
map (String _, String value):
  EmitIntermediate(extract_key(value), value);

reduce (String key, Iterator values):
  Emit(values);
```

Další příklady

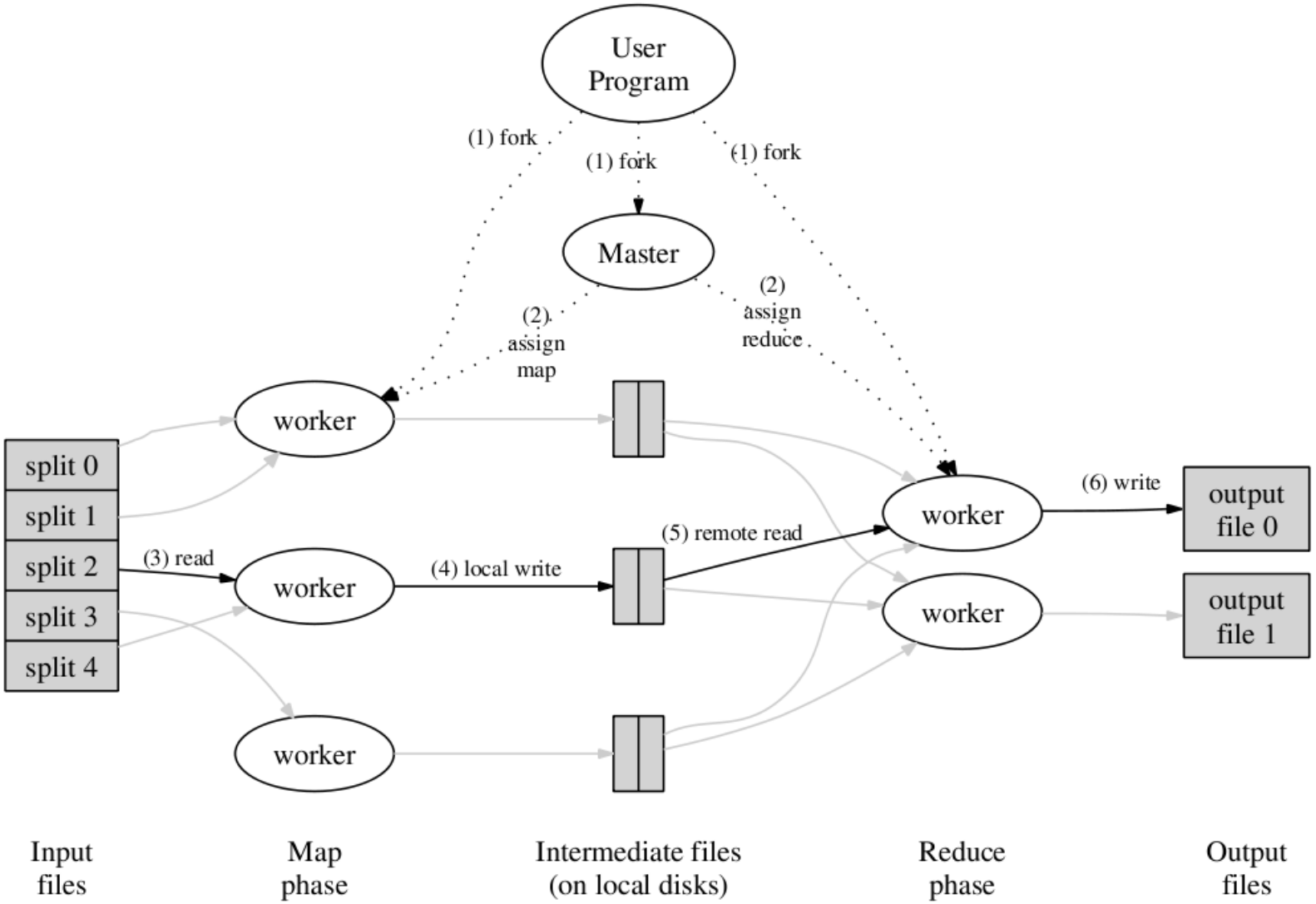
Invertovaný index

```
map (String docid, String content):
  for each word w in content:
    EmitIntermediate(word, docid);

reduce (String word, Iterator docids):
  Emit(sort(docids));
```

Třídění dat

```
map (String _, String value):
  EmitIntermediate(extract_key(value), value);

reduce (String key, Iterator values):
  Emit(values);
```

Přehled provádění



Průběh provádění

Průběh provádění:

1. Master získá `map`, `reduce`, data a počítače.
2. Vstup je rozdělen na M bloků.
3. `Map` na jednotlivé bloky dat.
4. Dočasné výsledky se rozdělí do R skupin.
5. `Reduce` na jednotlivé skupiny.

Průběh provádění

Průběh provádění:

1. Master získá `map`, `reduce`, data a počítače.
2. Vstup je rozdělen na M bloků.
3. `Map` na jednotlivé bloky dat.
4. Dočasné výsledky se rozdělí do R skupin.
5. `Reduce` na jednotlivé skupiny.

Průběh provádění

Průběh provádění:

1. Master získá `map`, `reduce`, data a počítače.
2. Vstup je rozdělen na M bloků.
3. `Map` na jednotlivé bloky dat.
4. Dočasné výsledky se rozdělí do R skupin.
5. `Reduce` na jednotlivé skupiny.

Průběh provádění

Průběh provádění:

1. Master získá `map`, `reduce`, data a počítače.
2. Vstup je rozdělen na M bloků.
3. `Map` na jednotlivé bloky dat.
4. Dočasné výsledky se rozdělí do R skupin.
5. `Reduce` na jednotlivé skupiny.

Průběh provádění

Průběh provádění:

1. Master získá `map`, `reduce`, data a počítače.
2. Vstup je rozdělen na M bloků.
3. `Map` na jednotlivé bloky dat.
4. Dočasné výsledky se rozdělí do R skupin.
5. `Reduce` na jednotlivé skupiny.

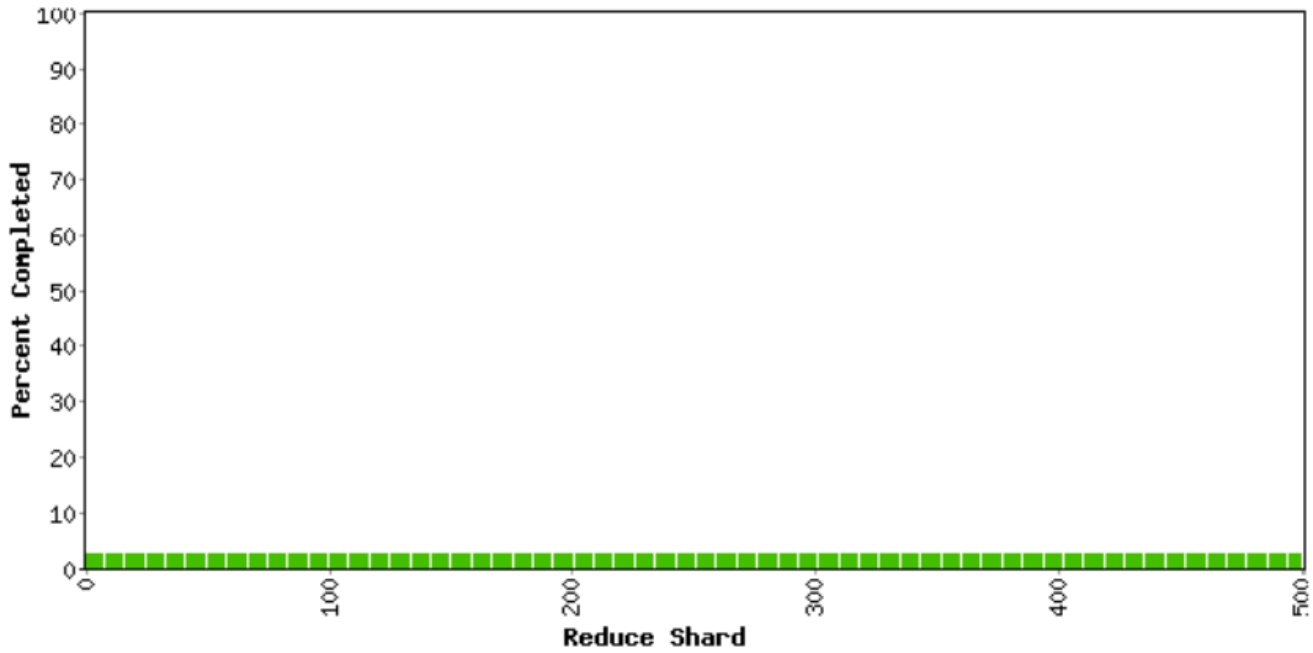
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

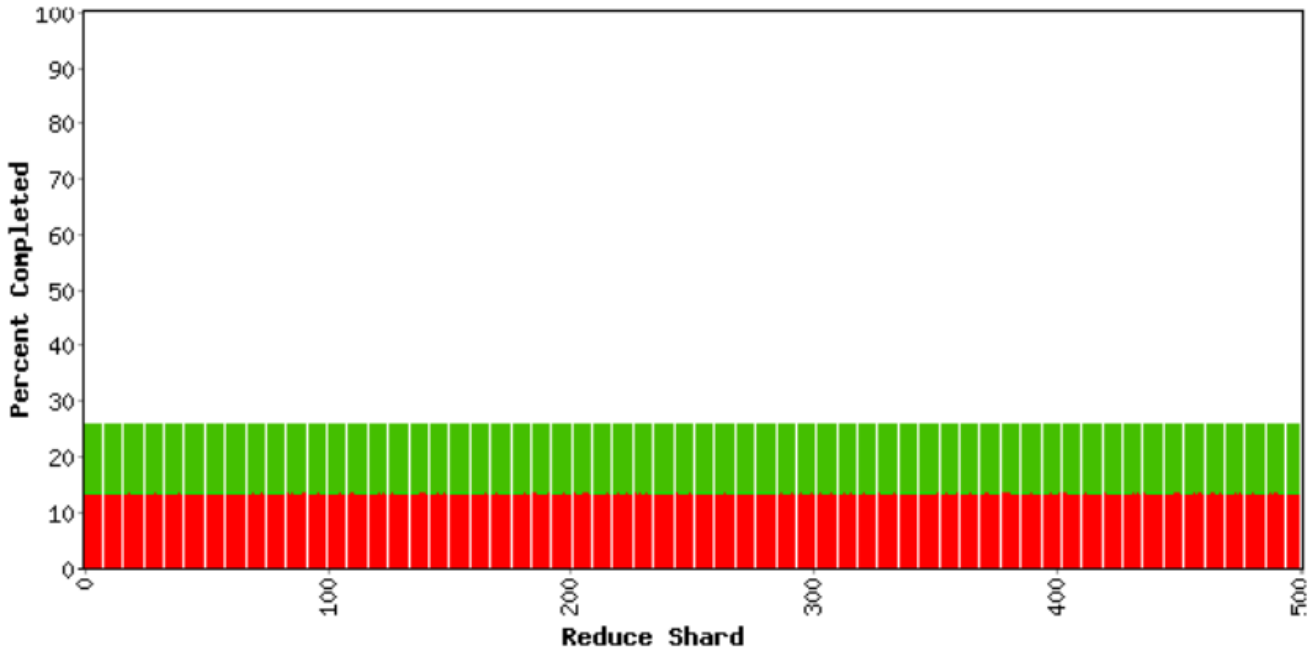
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

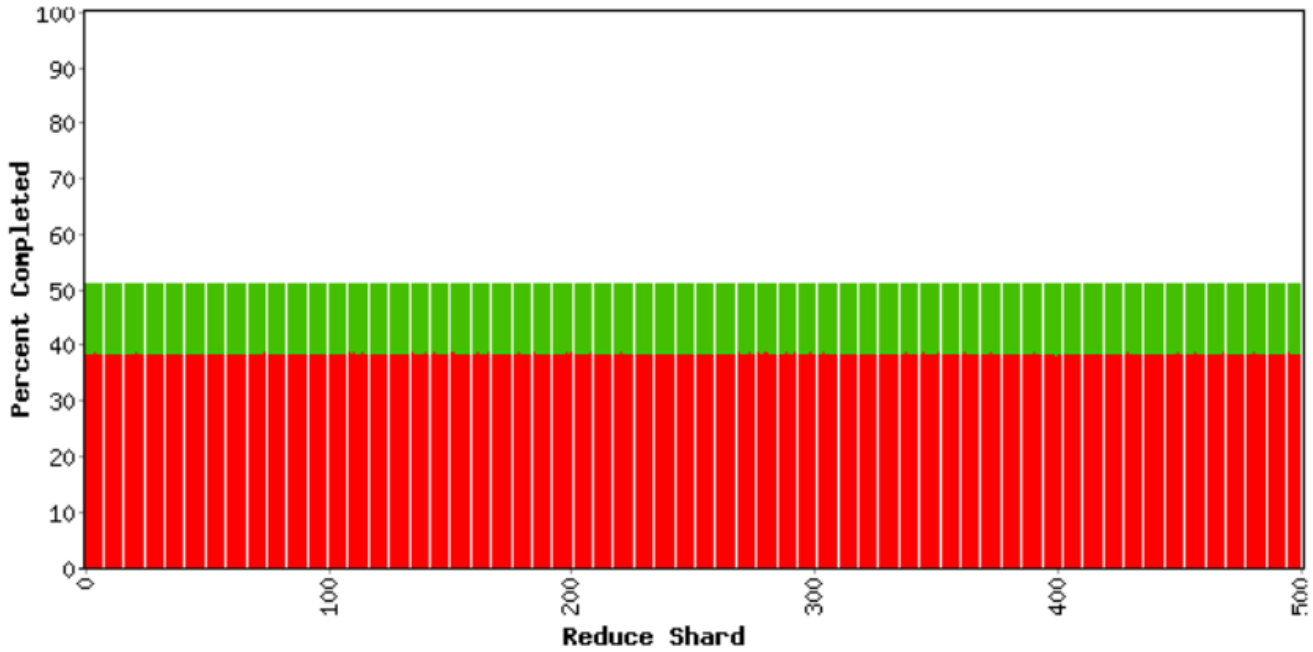
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

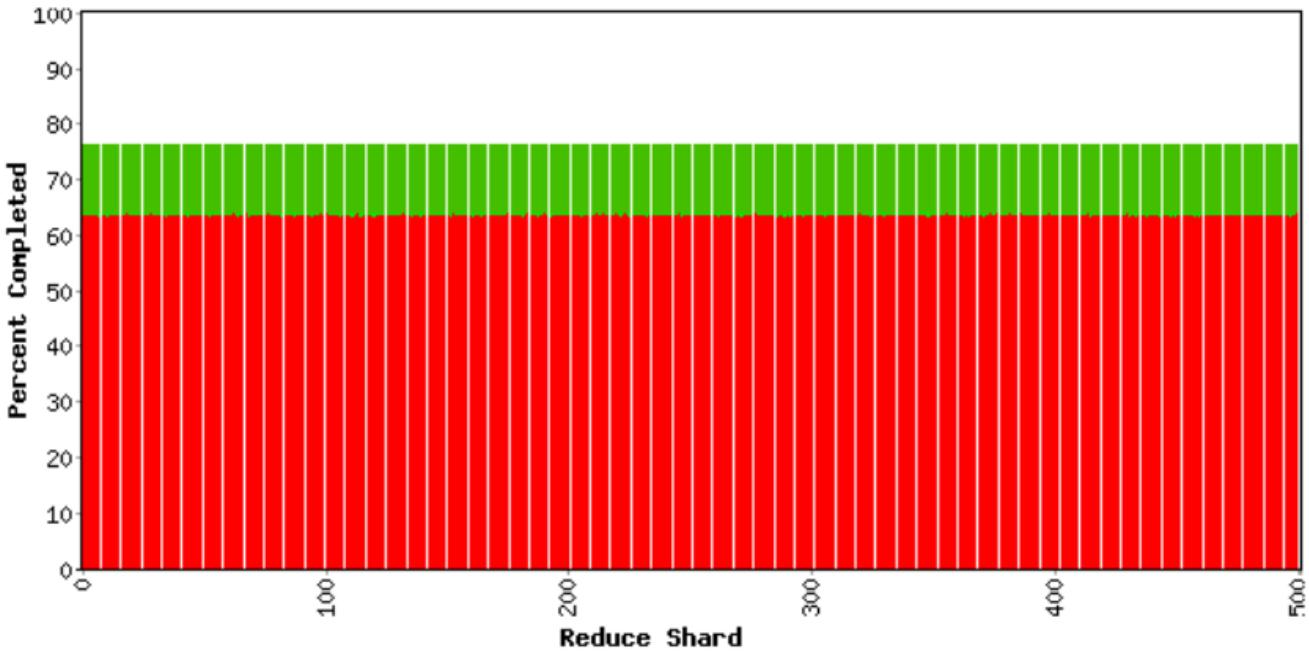
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

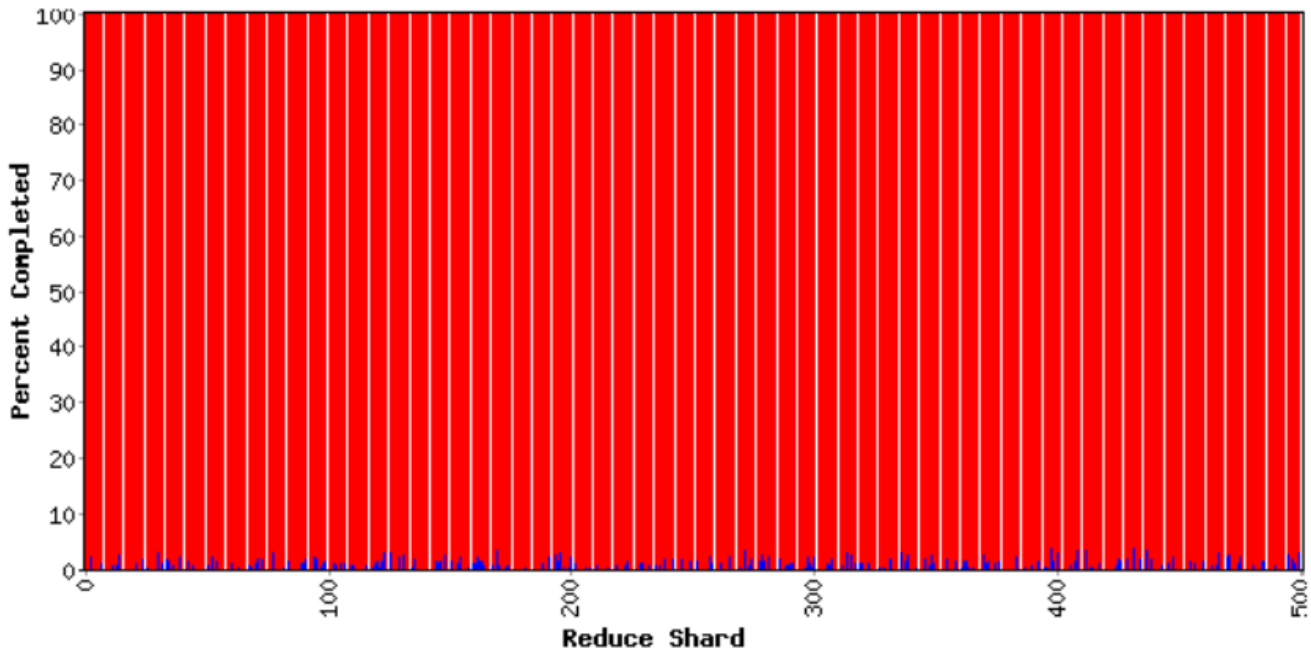
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

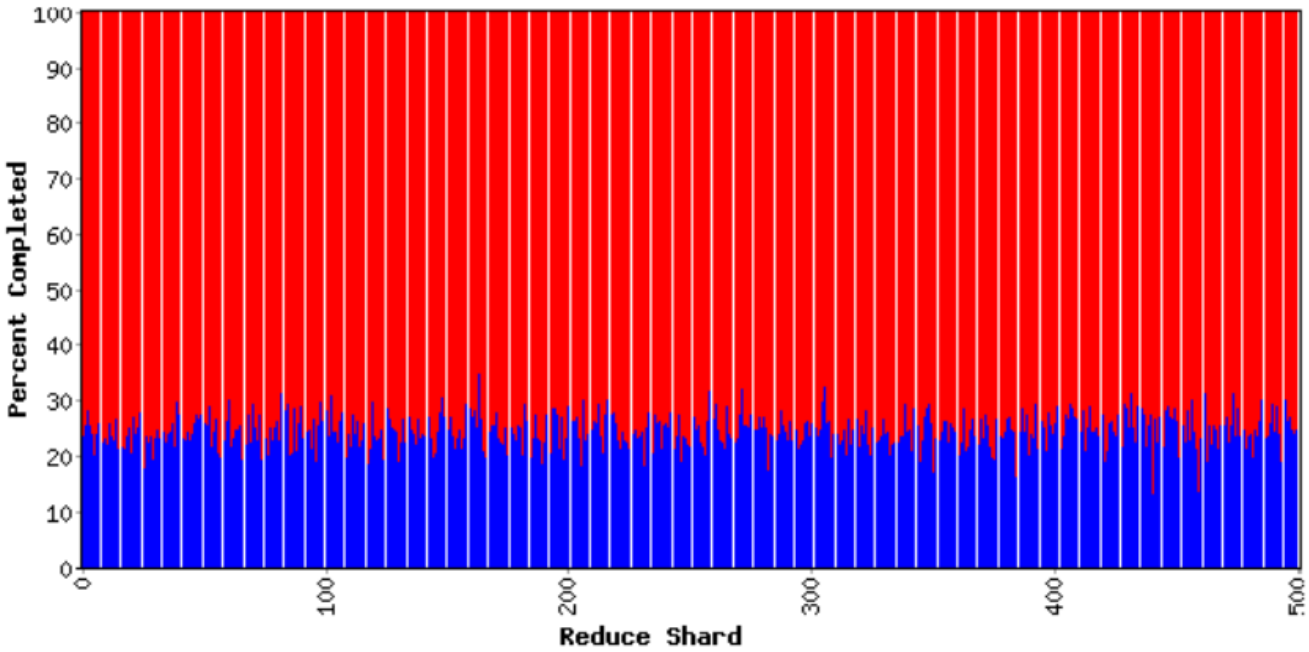
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.1	
Output (MB/s)	1238.8	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51738599	
mr-merge-outputs	51738599	

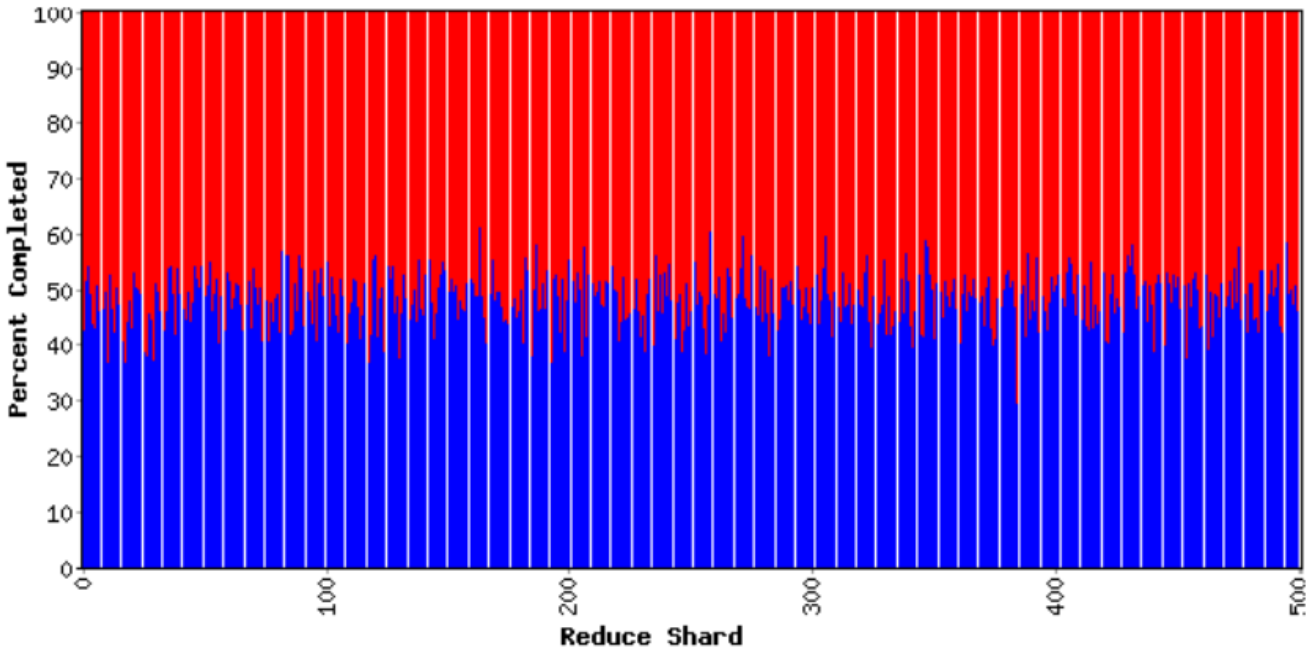
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

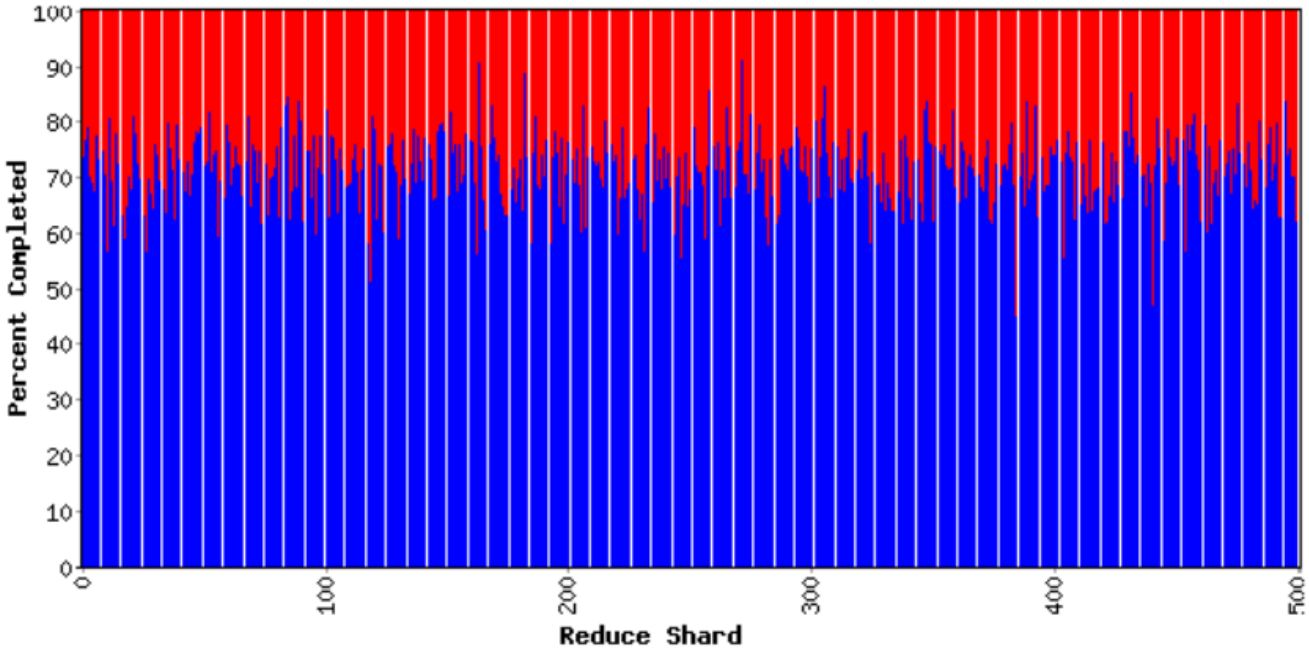
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1222.0	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51640600	
mr-merge-outputs	51640600	

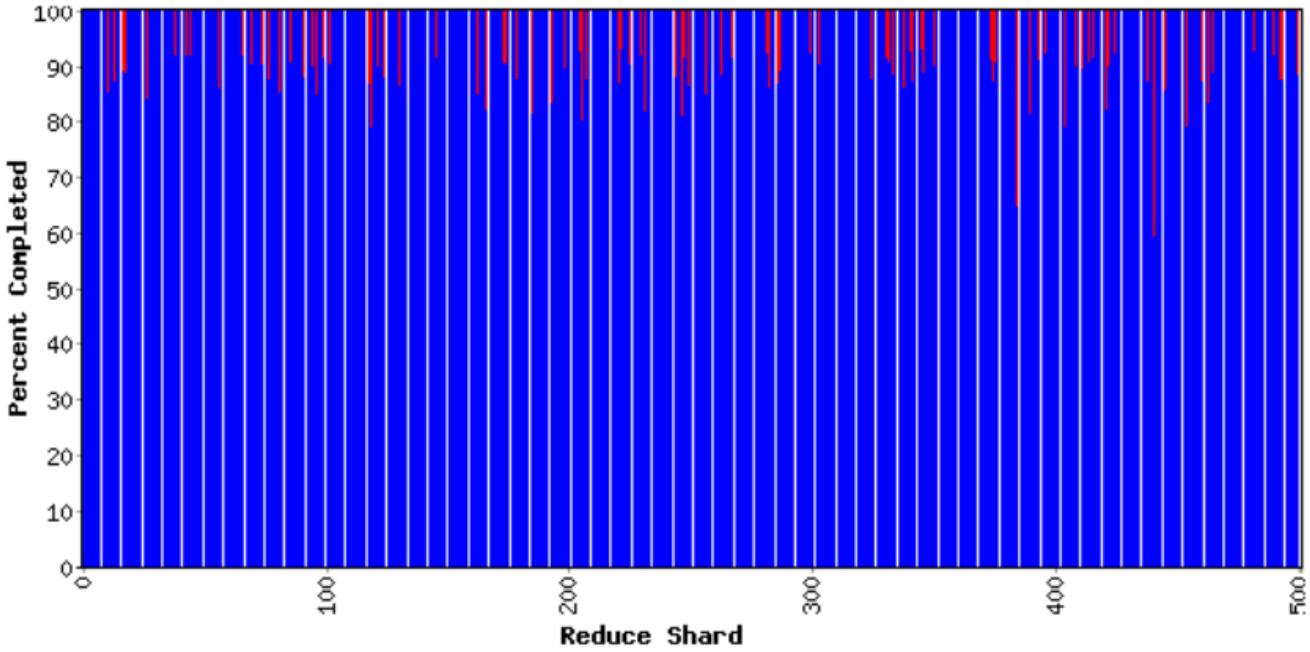
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	35083350	
mr-merge-outputs	35083350	

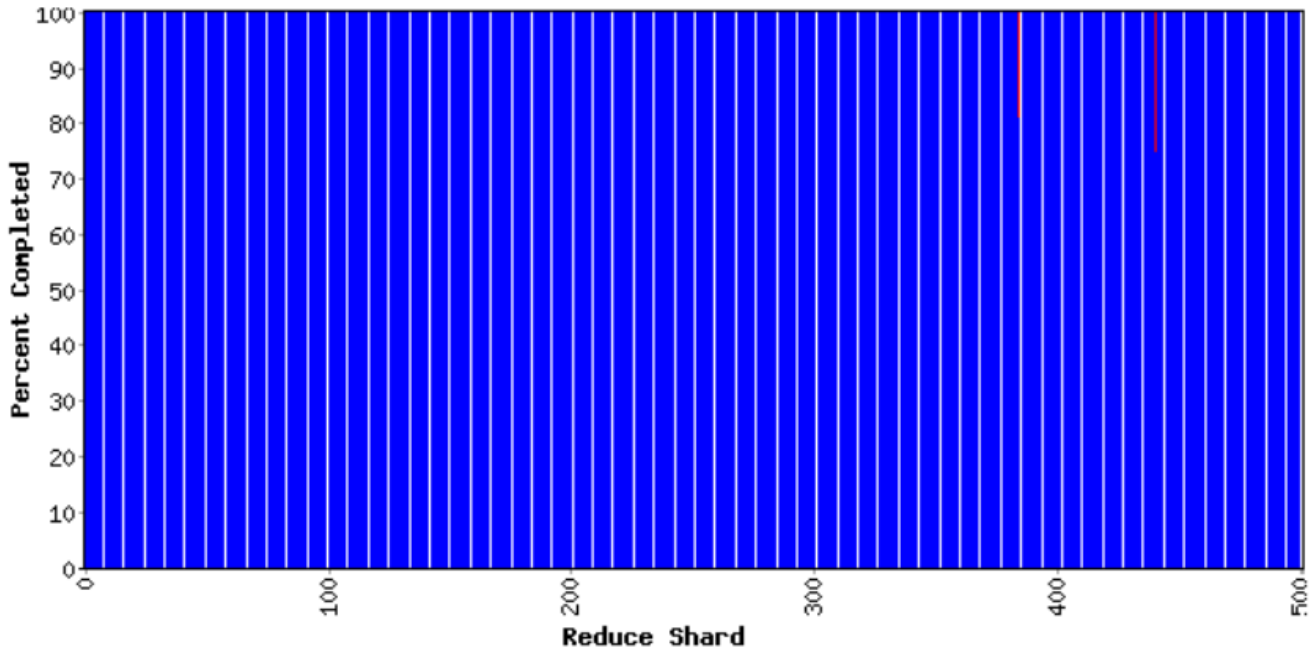
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	1
dups-in-index-merge	0	
mr-merge-calls	394792	1
mr-merge-outputs	394792	1

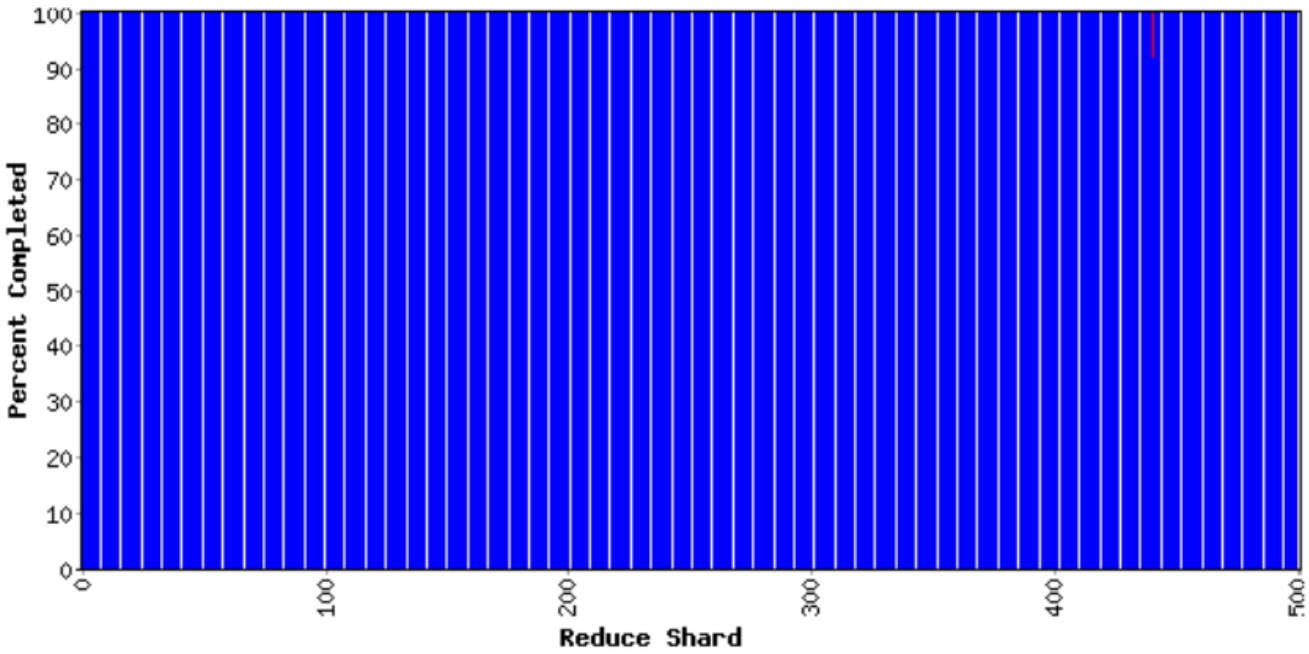
Sledování stavu výpočtu

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Detaily implementace

Detaily implementace:

1. Master a jeho datové struktury.
2. Lokalita zpracovávaných dat.
3. Volba M, R a vyrovnávání zátěže.
4. Výpadky použitých počítačů.
5. Synchronizace, zamykání a atomicita.
6. Záložní úlohy před koncem.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Rozšíření modelu

Model výpočtu může být rozšířen mnoha způsoby:

1. Vlastní rozdělovací funkce před `reduce` fází.
2. Zaručení pořadí dat při `reduce`.
3. Sekundární klíče.
4. Funkce `combine`.
5. Přeskakování problematických částí vstupu.
6. Vlastní čítače.
7. Lokální spouštění.
8. Stringové a strukturované rozhraní.

Zdrojový kód

Zveřejněný zdrojový kód:

```
// User's map function
class WordCounter : public Mapper {
 public:
  virtual void Map(const MapInput& input) {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
      // Skip past leading whitespace
      while ((i < n) && isspace(text[i]))
        i++;
      // Find word end
      int start = i;
      while ((i < n) && !isspace(text[i]))
        i++;
       if (start < i)
         Emit(text.substr(start,i-start),"1");
     }
   }
};
REGISTER_MAPPER(WordCounter);
```

```
// User's reduce function
class Adder : public Reducer {
 public:
   virtual void Reduce(ReduceInput* input) {
     // Iterate over all entries with the
     // same key and add the values
     int64 value = 0;
     while (!input->done()) {
       value += StringToInt(input->value());
       input->NextValue();
     }
     // Emit sum for input->key()
     Emit(IntToString(value));
   }
};
REGISTER_REDUCER(Adder);
```

Zdrojový kód

Zveřejněný zdrojový kód:

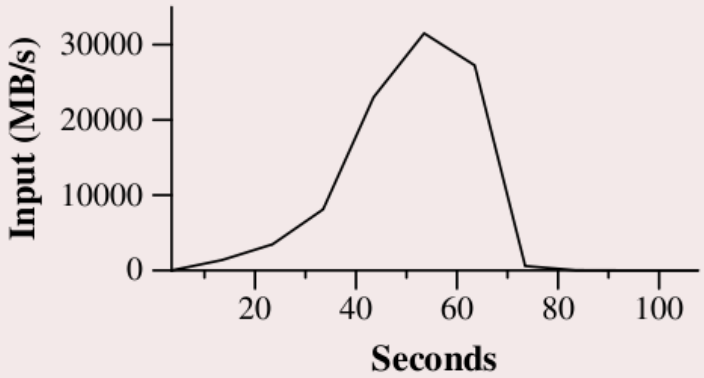
```
int main(int argc, char** argv) {
   ParseCommandLineFlags(argc, argv);
   MapReduceSpecification spec;
   // Store list of input files into "spec"
   for (int i = 1; i < argc; i++) {
     MapReduceInput* input = spec.add_input();
     input->set_format("text");
     input->set_filepattern(argv[i]);
     input->set_mapper_class("WordCounter");
   }

  // Specify the output files:
  //    /gfs/test/freq-00000-of-00100
  //    /gfs/test/freq-00001-of-00100
  //    ...
  MapReduceOutput* out = spec.output();
  out->set_filebase("/gfs/test/freq");
  out->set_num_tasks(100);
  out->set_format("text");
  out->set_reducer_class("Adder");
  // Optional: do partial sums within map
  // tasks to save network bandwidth
  out->set_combiner_class("Adder");
  // Tuning parameters: use at most 2000
  // machines and 100 MB of memory per task
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);
  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  // Done: 'result' structure contains info
  // about counters, time taken, number of
  // machines used, etc.
  return 0;
}
```

Výkon MapReduce

- Cluster s 1800 počítačí, každý dva 2GHz Xeony s HT, 4GB paměti, dva 160GB IDE disky, gigabitový Ethernet.

Hledání záznamů

Celkem 10^{10} 100B záznamů, $M = 15000$ (64MB kusy), $R = 1$.

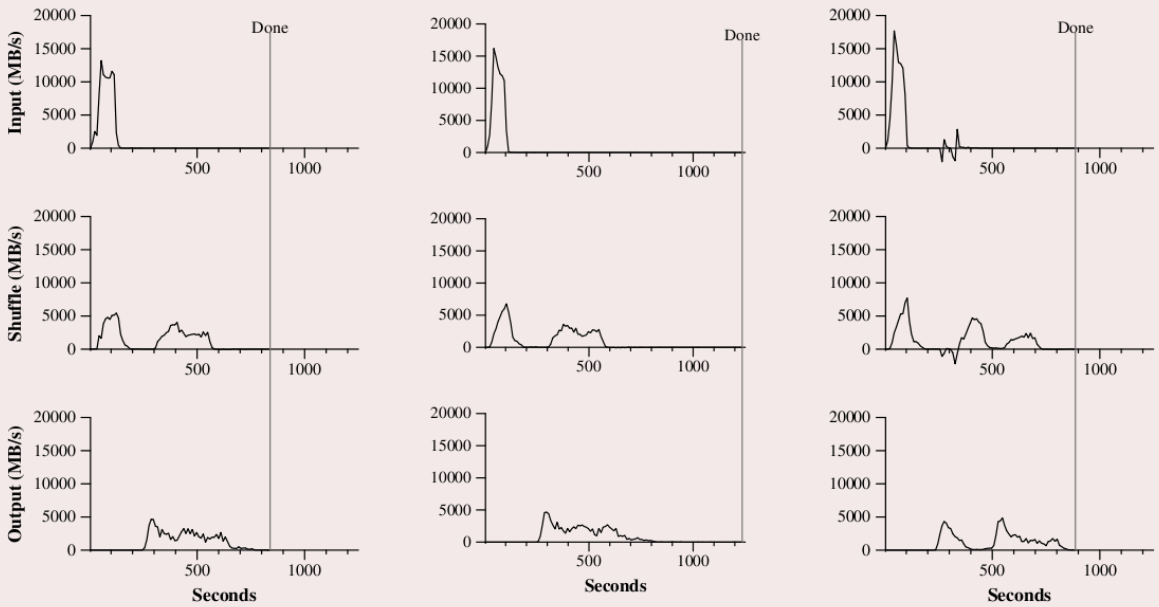


Výkon MapReduce

- Cluster s 1800 počítačí, každý dva 2GHz Xeony s HT, 4GB paměti, dva 160GB IDE disky, gigabitový Ethernet.

Třídění

Celkem 10^{10} 100B záznamů, $M = 15000$ (64MB kusy), $R = 1$.



(a) Normal execution (b) No backup tasks (c) 200 tasks killed

Používání MapReduce uvnitř firmy Google

MapReduce spuštěné ve firmě Google v srpnu 2004:

Number of jobs	29,423
Average job completion time	634 secs
Machine days used	79,186 days
Input data read	3,288 TB
Intermediate data produced	758 TB
Output data written	193 TB
Average worker machines per job	157
Average worker deaths per job	1.2
Average map tasks per job	3,351
Average reduce tasks per job	55
Unique *map* implementations	395
Unique *reduce* implementations	269
Unique *map/reduce* combinations	426

Výhody a nevýhody modelu MapReduce

Výhody MapReduce

1. jednoduchost
2. vysoká úroveň paralelismu
3. veliká škálovatelnost
4. nenáročnost na HW

Vhodné typy problémů pro MapReduce

1. vytváření indexů
2. strojové učení na velkých datech, například automatický překlad
3. extrahování dat
4. statistiky a analýza dat
5. clusterizace
6. manipulace s velkými grafy

Výhody a nevýhody modelu MapReduce

Výhody MapReduce

1. jednoduchost
2. vysoká úroveň paralelismu
3. veliká škálovatelnost
4. nenáročnost na HW

Vhodné typy problémů pro MapReduce

1. vytváření indexů
2. strojové učení na velkých datech, například automatický překlad
3. extrahování dat
4. statistiky a analýza dat
5. clusterizace
6. manipulace s velkými grafy

Hadoop

Hadoop

1. Opensource implementace MapReduce Hadoop.
2. Opensource implementace GFS – HDFS.
3. Zastřešuje Apache, pod Apache licencí.
4. V jazyce Java. . .
5. . . . ale `map` a `reduce` lze psát v mnoha jazycích.

Hadoop

Hadoop

1. Opensource implementace MapReduce Hadoop.
2. Opensource implementace GFS – HDFS.
3. Zastřešuje Apache, pod Apache licencí.
4. V jazyce Java. . .
5. . . . ale `map` a `reduce` lze psát v mnoha jazycích.

Hadoop

Hadoop

1. Opensource implementace MapReduce Hadoop.
2. Opensource implementace GFS – HDFS.
3. Zastřešuje Apache, pod Apache licencí.
4. V jazyce Java. . .
5. . . . ale `map` a `reduce` lze psát v mnoha jazycích.

Hadoop

Hadoop

1. Opensource implementace MapReduce Hadoop.
2. Opensource implementace GFS – HDFS.
3. Zastřešuje Apache, pod Apache licencí.
4. V jazyce Java. . .
5. . . . ale `map` a `reduce` lze psát v mnoha jazycích.

Hadoop

Hadoop

1. Opensource implementace MapReduce Hadoop.
2. Opensource implementace GFS – HDFS.
3. Zastřešuje Apache, pod Apache licencí.
4. V jazyce Java. . .
5. . . . ale `map` a `reduce` lze psát v mnoha jazycích.

Kdo používá Hadoop

Kdo používá Hadoop:

- Yahoo, $\sim$ 100 000 procesorů v $\sim$ 20 000 počítačích
- Quantcast, $\sim$ 3 000 procesorů, 3.5PB dat
- Facebook, $\sim$ 2 560 procesorů v $\sim$ 320 počítačích, 1.3PB dat
- IBM, dává univerzitám přístup k Hadoop clusterům
- ImageShack, Last.fm, The New York Times, . . .

Zdrojový kód

```
public class WordCount {

   public static class Map extends MapReduceBase
                            implements Mapper<LongWritable, Text, Text, IntWritable> {
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();

     public void map(LongWritable key, Text value,
                     OutputCollector<Text, IntWritable> output, Reporter reporter)
                                                                  throws IOException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         output.collect(word, one);
       }
     }
   }

   public static class Reduce extends MapReduceBase
                               implements Reducer<Text, IntWritable, Text, IntWritable> {
     public void reduce(Text key, Iterator<IntWritable> values,
                        OutputCollector<Text, IntWritable> output, Reporter reporter)
                                                                  throws IOException {
       int sum = 0;
       while (values.hasNext()) {
         sum += values.next().get();
       }
       output.collect(key, new IntWritable(sum));
     }
   }
```

Zdrojový kód

```
    public static void main(String[] args) throws Exception {
      JobConf conf = new JobConf(WordCount.class);
      conf.setJobName(”wordcount”);

      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);

      conf.setMapperClass(Map.class);
      conf.setCombinerClass(Reduce.class);
      conf.setReducerClass(Reduce.class);

      conf.setInputFormat(TextInputFormat.class);
      conf.setOutputFormat(TextOutputFormat.class);

      FileInputFormat.setInputPaths(conf, new Path(args[0]));
      FileOutputFormat.setOutputPath(conf, new Path(args[1]));

      JobClient.runJob(conf);
    }
}
```

Zdrojový kód - Hadoop Pipes

```
#include ”hadoop/Pipes.hh”
#include ”hadoop/TemplateFactory.hh”
#include ”hadoop/StringUtils.hh”

const std::string WORDCOUNT = ”WORDCOUNT”;
const std::string INPUT_WORDS = ”INPUT_WORDS”;
const std::string OUTPUT_WORDS = ”OUTPUT_WORDS”;

class WordCountMap: public HadoopPipes::Mapper {
public:
  HadoopPipes::TaskContext::Counter* inputWords;

  WordCountMap(HadoopPipes::TaskContext& context) {
    inputWords = context.getCounter(WORDCOUNT, INPUT_WORDS);
  }

  void map(HadoopPipes::MapContext& context) {
    std::vector<std::string> words =
      HadoopUtils::splitString(context.getInputValue(), ” ”);
    for(unsigned int i=0; i < words.size(); ++i) {
      context.emit(words[i], ”1”);
    }
    context.incrementCounter(inputWords, words.size());
  }
};
```

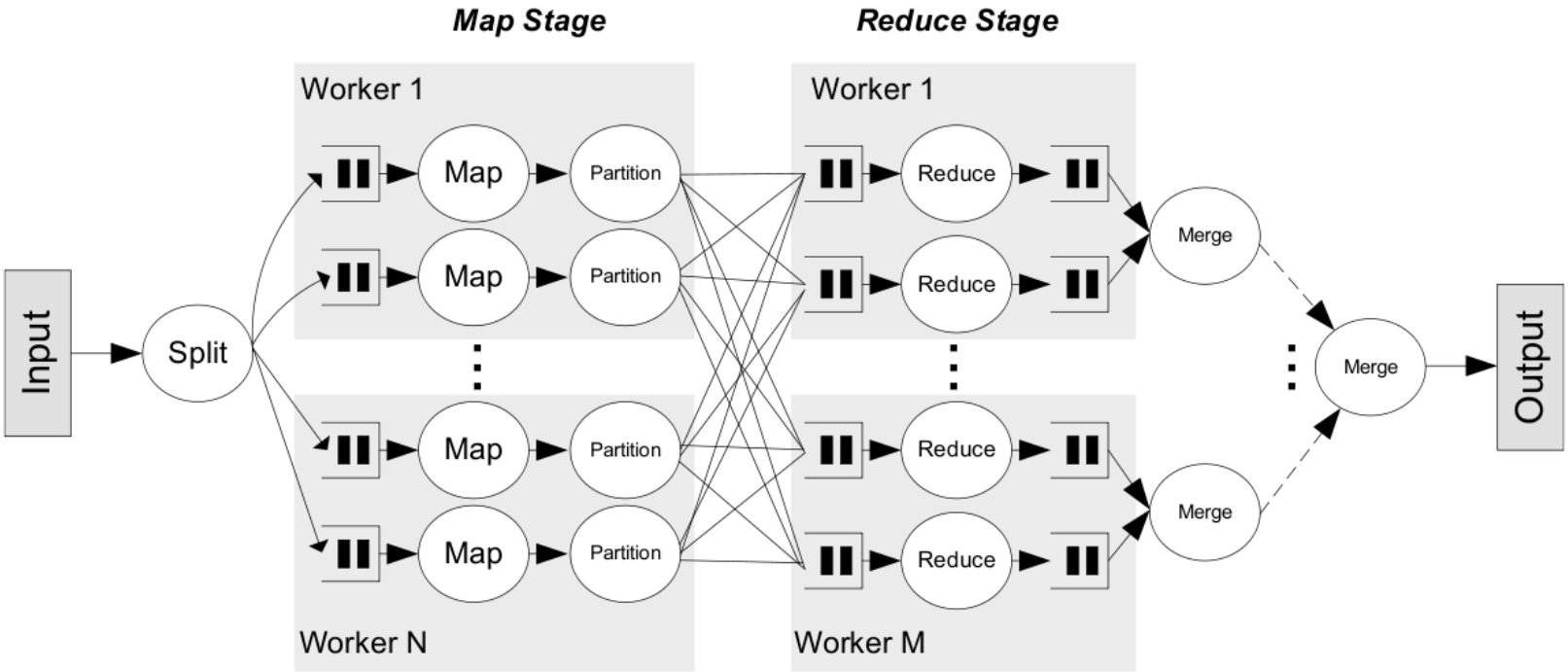
Zdrojový kód - Hadoop Pipes

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
  HadoopPipes::TaskContext::Counter* outputWords;

  WordCountReduce(HadoopPipes::TaskContext& context) {
    outputWords = context.getCounter(WORDCOUNT, OUTPUT_WORDS);
  }

  void reduce(HadoopPipes::ReduceContext& context) {
    int sum = 0;
    while (context.nextValue()) {
      sum += HadoopUtils::toInt(context.getInputValue());
    }
    context.emit(context.getInputKey(), HadoopUtils::toString(sum));
    context.incrementCounter(outputWords, 1);
  }
};

class WordCountPartitioner: public HadoopPipes::Partitioner {
public:
  WordCountPartitioner(HadoopPipes::TaskContext& context){}
  virtual int partition(const std::string& key, int numOfReduces) {
    return 0;
  }
};

int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(HadoopPipes::TemplateFactory<WordCountMap,
                              WordCountReduce,WordCountPartitioner,
                              WordCountReduce>());
}
```

Phoenix

- Implementace MapReduce v prostředí se sdílenou pamětí.

Přehled zpracování dat ve Phoenixu



Phoenix API

- `int phoenix_scheduler(scheduler args t * args)`
- `void emit_intermediate(void *key, void *val, int key size)`
- `void emit(void *key, void *val)`
- `int (*splitter_t)(void *, int, map args t *)`
- `void (*map_t)(map args t*)`
- `int (*partition_t)(int, void *, int)`
- `void (*reduce_t)(void *, void **, int)`
- `int (*key_cmp_t)(const void *, const void*)`

Phoenix API

- `int phoenix_scheduler(scheduler args t * args)`
- `void emit_intermediate(void *key, void *val, int key size)`
- `void emit(void *key, void *val)`
- `int (*splitter_t)(void *, int, map args t *)`
- `void (*map_t)(map args t*)`
- `int (*partition_t)(int, void *, int)`
- `void (*reduce_t)(void *, void **, int)`
- `int (*key_cmp_t)(const void *, const void*)`

Detaily implementace Phoenixu

1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Detaily implementace Phoenixu

1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Detaily implementace Phoenixu

1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Detaily implementace Phoenixu

1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Detaily implementace Phoenixu

1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Detaily implementace Phoenixu

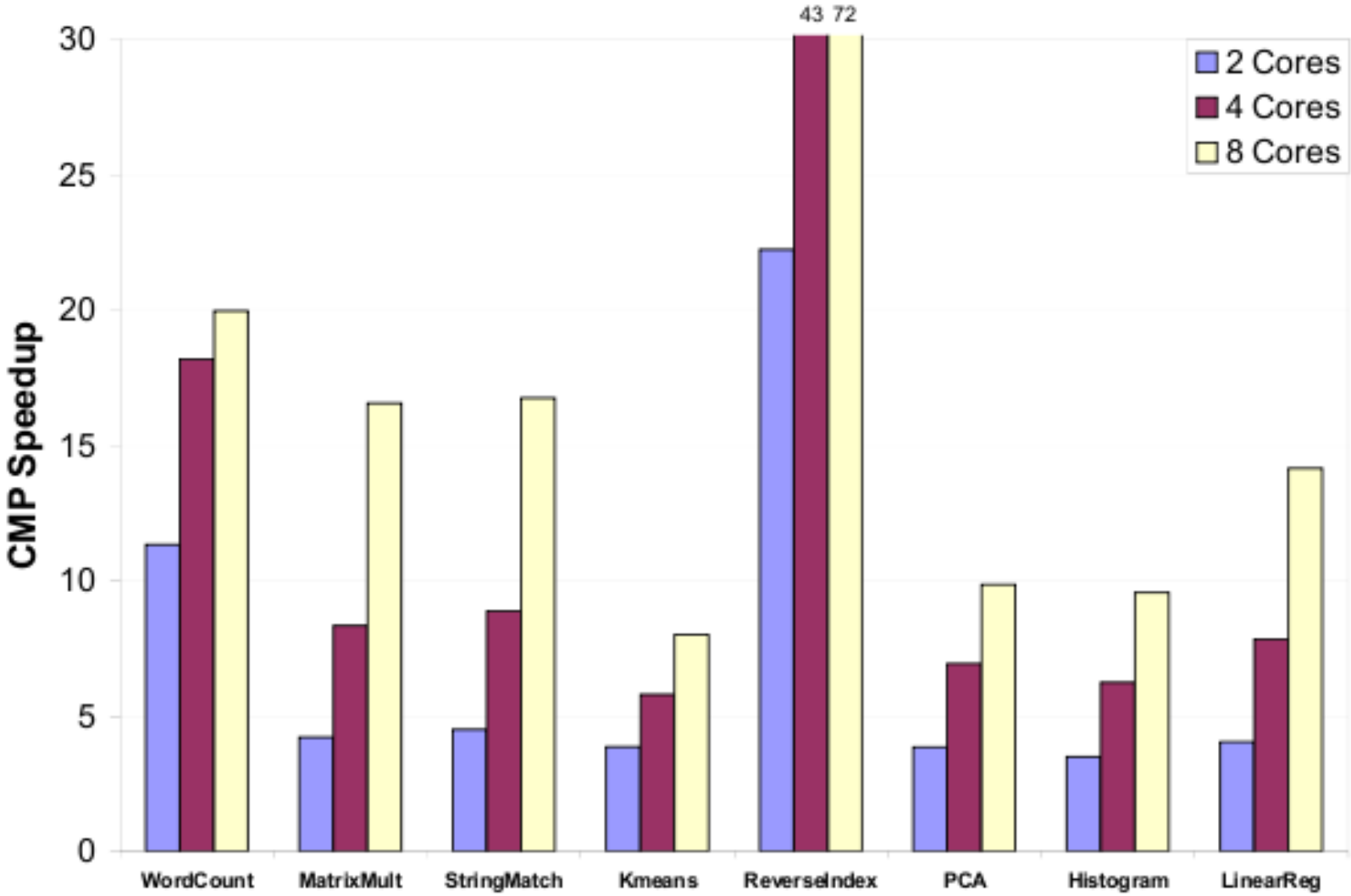
1. Předávání dat, dočasné buffery.
2. Dynamické přidělování úkolů.
3. Počty souběžně běžících vláken / procesů.
4. Velikosti vstupů pro `map` řádově jako L1 cache.
5. Detekce chyb, záložní úlohy před koncem.

Výkon Phoenixu

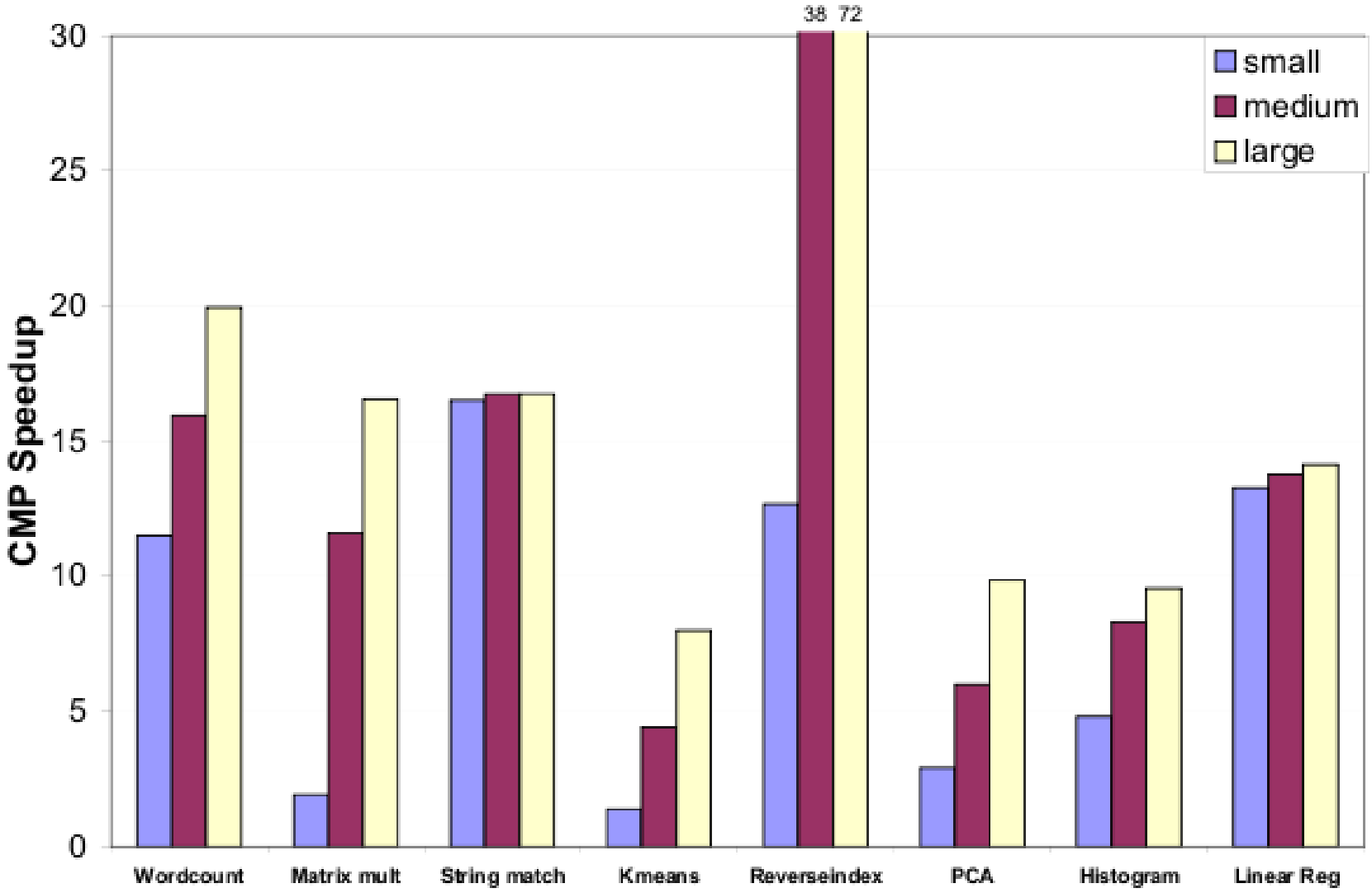
	CMP	SMP
Model	Sun Fire T1200	Sun Ultra-Enterprise 6000
CPU Type	UltraSparc T1 single-issue in-order	UltraSparc II 4-way issue in-order
CPU Count	8	24
Threads/CPU	4	1
L1 Cache	8KB 4-way SA	16KB DM
L2 Size	3MB 12-way SA shared	512KB per CPU (off chip)
Clock Freq.	1.2 GHz	250 MHz

	Description	Data Sets	Code Size Ratio	
			Pthreads	Phoenix
Word Count	Determine frequency of words in a file	S:10MB, M:50MB, L:100MB	1.8	0.9
Matrix Multiply	Dense integer matrix multiplication	S:100x100, M:500x500, L:1000x1000	1.8	2.2
Reverse Index	Build reverse index for links in HTML files	S:100MB, M:500MB, L:1GB	1.5	0.9
Kmeans	Iterative clustering algorithm to classify 3D data points into groups	S:10K, M:50K, L:100K points	1.2	1.7
String Match	Search file with keys for an encrypted word	S:50MB, M:100MB, L:500MB	1.8	1.5
PCA	Principal components analysis on a matrix	S:500x500, M:1000x1000, L:1500x1500	1.7	2.5
Histogram	Determine frequency of each RGB component in a set of images	S:100MB, M:400MB, L:1.4GB	2.4	2.2
Linear Regression	Compute the best fit line for a set of points	S:50M, M:100M, L:500M	1.7	1.6

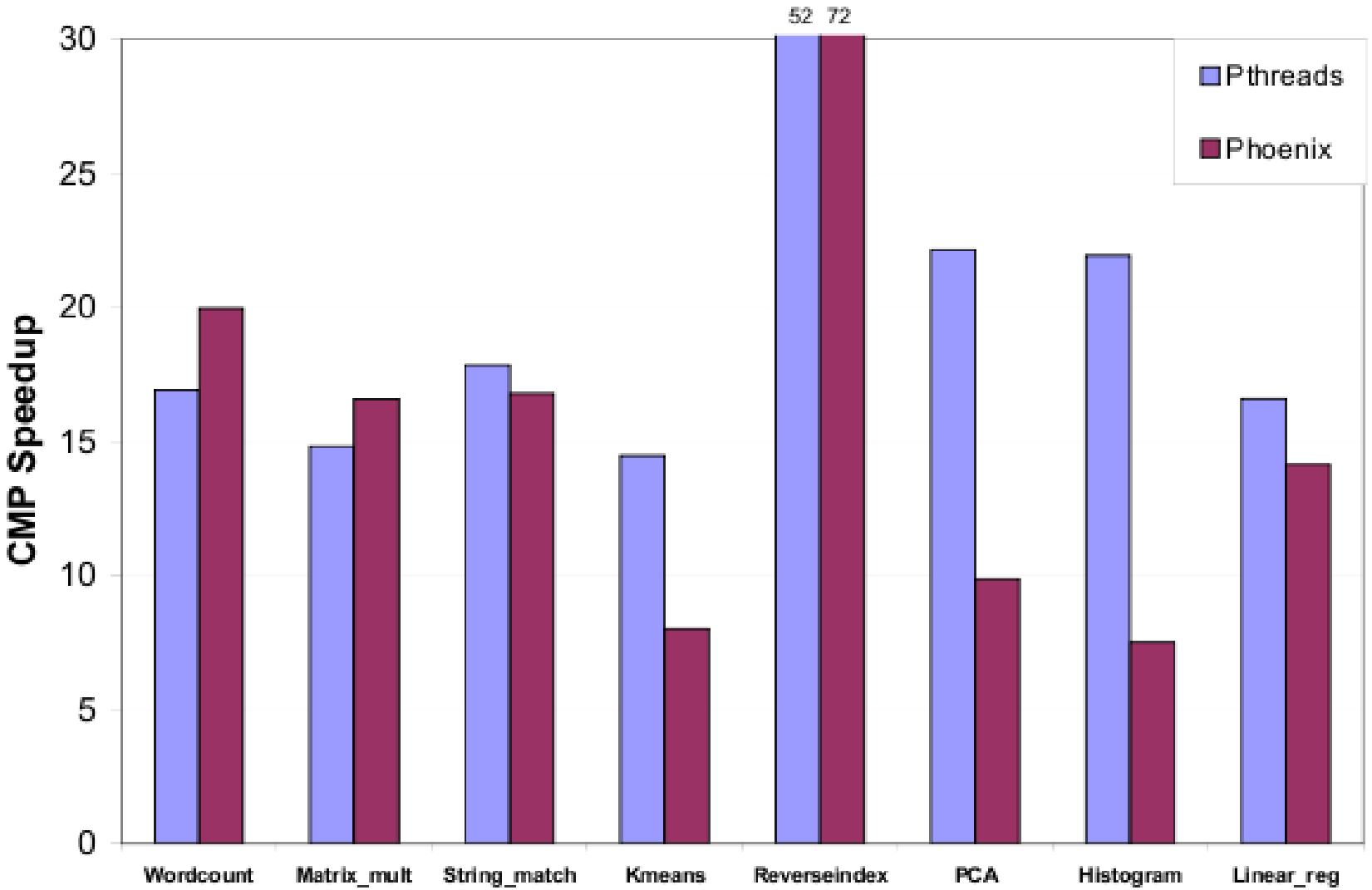
Výkon Phoenixu



Výkon Phoenixu



Výkon Phoenixu



Mars

Implementace MapReduce v prostředí se sdílenou pamětí pomocí GPU.

Detaily implementace Marsu

1. Velmi pobobné Phoenixu, rozdíly pouze kvůli vlastnostem GPU.
2. Základní API:
   - `void map_count(void *key, void *val, int keySize, int valSize)`
   - `void map(void *key, void* val, int keySize, int valSize)`
   - `void reduce_count(void* key, void* vals, int keySize, int valCount)`
   - `void reduce(void* key, void* vals, int keySize, int count)`
   - `void emit_intermediate_count(int keySize, int valSize)`
   - `void emit_intermediate(void* key, void* val, int keySize, int valSize)`
   - `void emit_count(int keySize, int valSize)`
   - `void emit(void *key, void* val, int keySize, int valSize)`

Detaily implementace Marsu

1. Velmi pobobné Phoenixu, rozdíly pouze kvůli vlastnostem GPU.
2. Základní API:
   - `void map_count(void *key, void *val, int keySize, int valSize)`
   - `void map(void *key, void* val, int keySize, int valSize)`
   - `void reduce_count(void* key, void* vals, int keySize, int valCount)`
   - `void reduce(void* key, void* vals, int keySize, int count)`
   - `void emit_intermediate_count(int keySize, int valSize)`
   - `void emit_intermediate(void* key, void* val, int keySize, int valSize)`
   - `void emit_count(int keySize, int valSize)`
   - `void emit(void *key, void* val, int keySize, int valSize)`

Detaily implementace Marsu

1. Velmi pobobné Phoenixu, rozdíly pouze kvůli vlastnostem GPU.
2. Základní API:
   - `void map_count(void *key, void *val, int keySize, int valSize)`
   - `void map(void *key, void* val, int keySize, int valSize)`
   - `void reduce_count(void* key, void* vals, int keySize, int valCount)`
   - `void reduce(void* key, void* vals, int keySize, int count)`
   - `void emit_intermediate_count(int keySize, int valSize)`
   - `void emit_intermediate(void* key, void* val, int keySize, int valSize)`
   - `void emit_count(int keySize, int valSize)`
   - `void emit(void *key, void* val, int keySize, int valSize)`

Přehled zpracování dat v Marsu

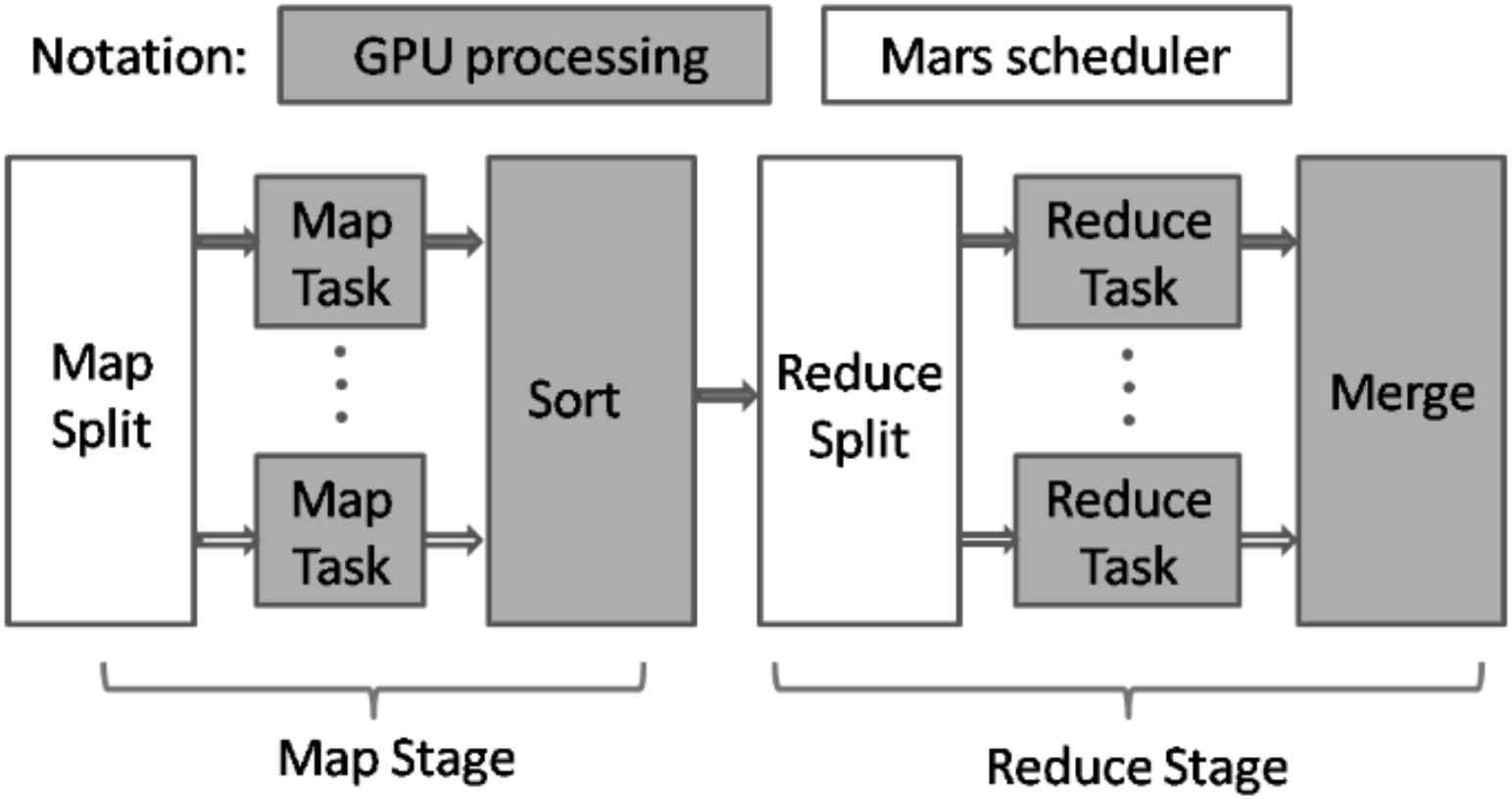


Figure 2. The work flow of Mars on the GPU.

Testovací počítače a úlohy

	GPU	CPU
Processors	1350MHz × 8 × 16	2.4 GHz × 4
Data cache (shared memory)	16KB × 16	L1: 32KB × 4, L2: 4096KB × 2
Cache latency (cycle)	2	L1: 2 , L2: 8
DRAM (MB)	768	2048
DRAM latency (cycle)	200	300
Bus width (bit)	384	64
Memory clock (GHz)	1.8	1.3

App.	Description	Data sets
String Match	Find the position of a string in a file.	S: 32MB, M: 64 MB, L: 128 MB
Inverted Index	Build inverted index for links in HTML files.	S: 16MB, M: 32 MB, L: 64MB
Similarity Score	Compute the pair-wise similarity score for a set of documents.	#doc: S: 512, M: 1024, L: 2048. #feature dimension: 128.
Matrix Multiplication	Multiply two matrices.	#dimension: S: 512, M: 1024, L: 2048
Page View Count	Count the page view number of each URL in the web log.	S: 32MB, M: 64 MB, L: 96 MB
Page View Rank	Find the top-10 hot pages in the web log.	S: 32MB, M: 64 MB, L: 96 MB

Výkon Marsu v porovnání s Phoenixem

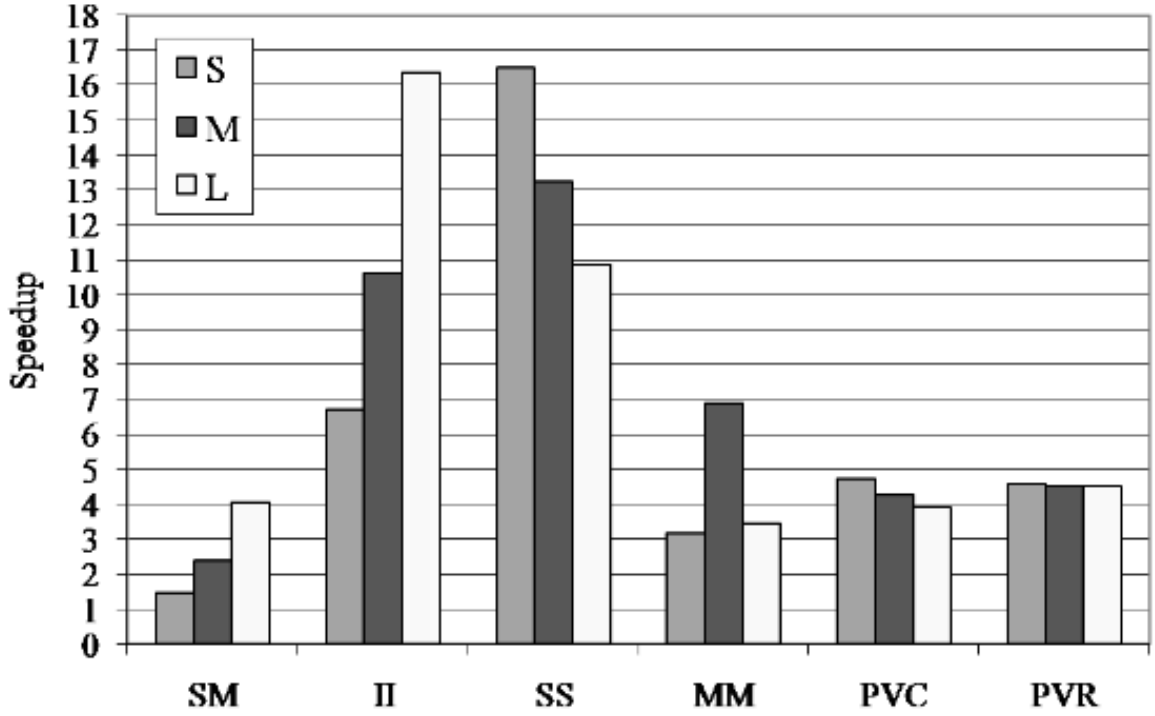


Figure 6. Performance speedup of Mars over Phoenix.

Přehled zpracování dat v Marsu++

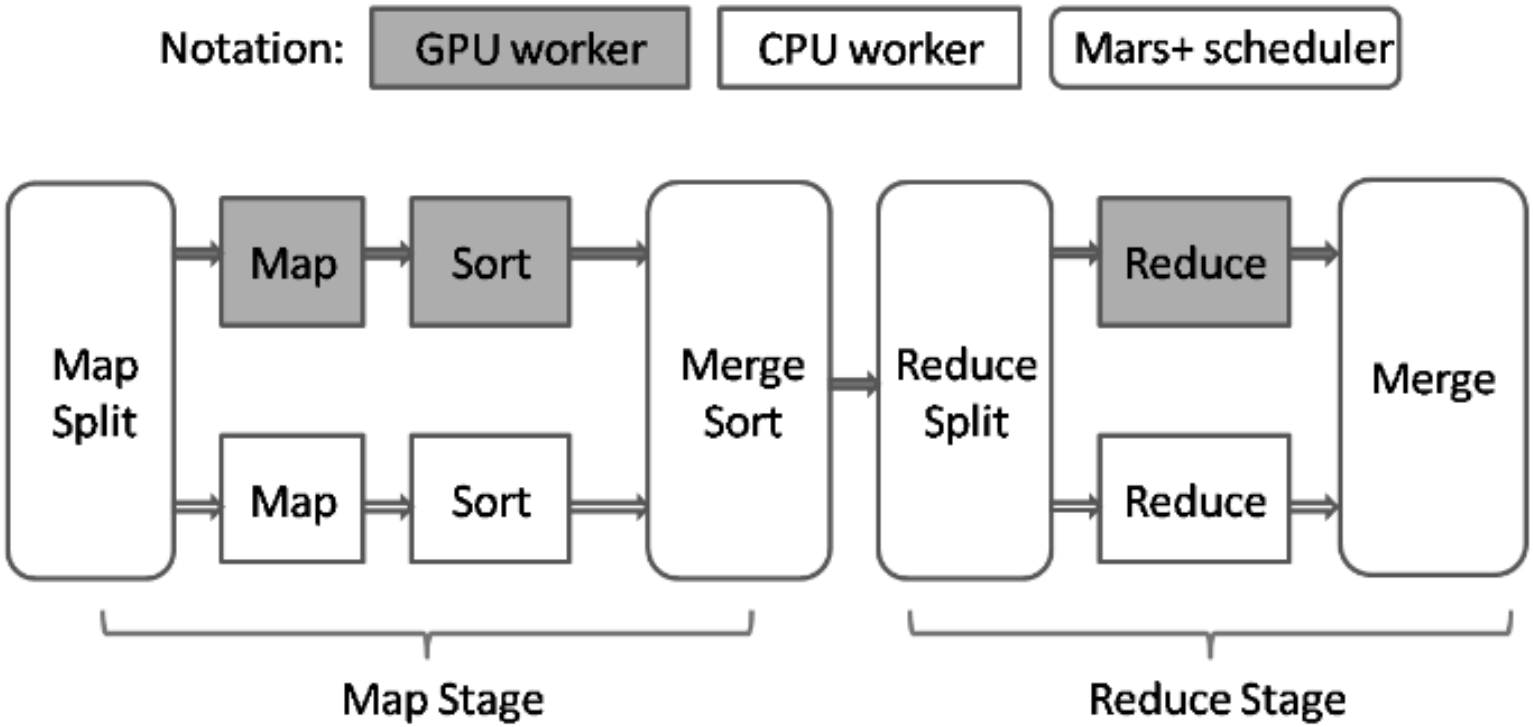


Figure 4. The work flow of Mars+.

Výkon Marsu++ v porovnání s Marsem

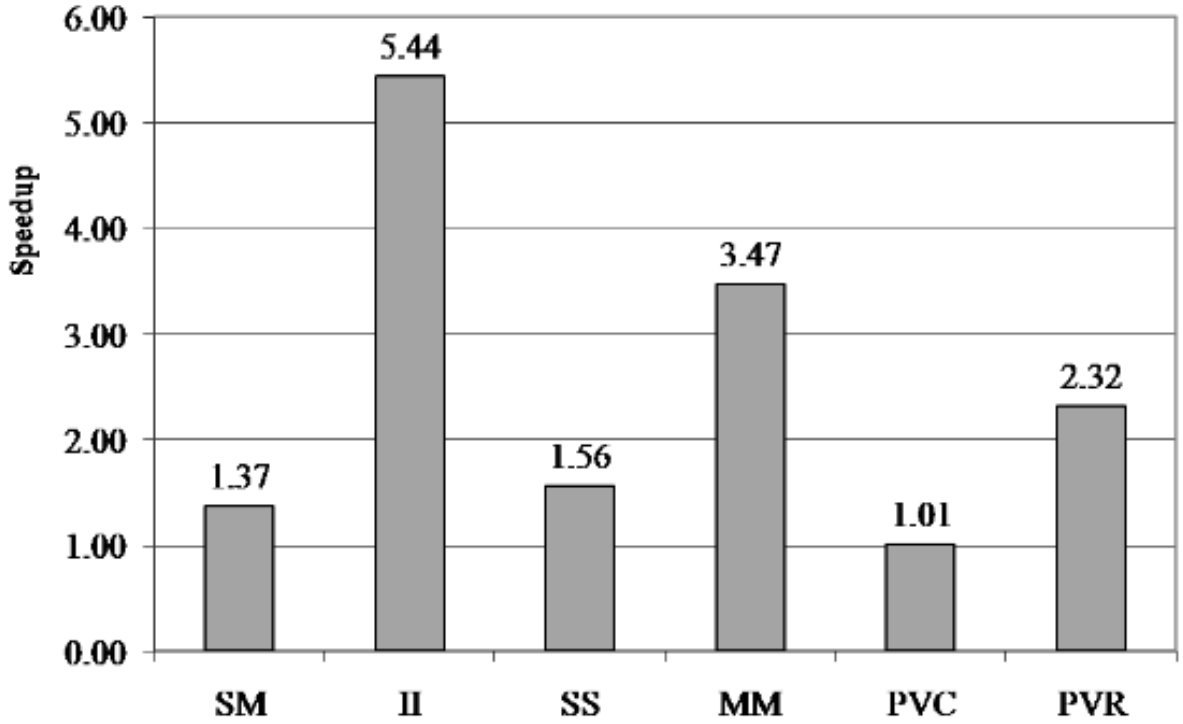


Figure 11. Performance speedup of the CPU worker in Mars+ over Phoenix.

Otázky?