MapReduce for Large Scale Computing

Rationale for Map-Reduce

Map-reduce idea: An Example (part 1)

This example uses JavaScript.

```
// A trivial example:
  alert("I'd like some Spaghetti!");
  alert("I'd like some Chocolate Mousse!");
```

The above can be improved to the following code:

Map-reduce idea: An Example (part 1)

This example uses JavaScript.

```
// A trivial example:
  alert("I'd like some Spaghetti!");
  alert("I'd like some Chocolate Mousse!");
```

The above can be improved to the following code:

```
function SwedishChef( food )
{
    alert("I'd like some " + food + "!");
}

SwedishChef("Spaghetti");
SwedishChef("Chocolate Mousse");
```

Map-reduce idea: An Example (part 2)

```
alert("get the lobster");
PutInPot("lobster");
PutInPot("water");

alert("get the chicken");
BoomBoom("chicken");
BoomBoom("coconut");
```

The above can be improved to the following code using function pointers!

Map-reduce idea: An Example (part 2)

```
    alert("get the lobster");
    PutInPot("lobster");
    PutInPot("water");

    alert("get the chicken");
    BoomBoom("chicken");
    BoomBoom("coconut");
```

The above can be improved to the following code using function pointers!

```
function Cook( i1, i2, f )
{
    alert("get the " + i1);
    f(i1);
    f(i2);
}

Cook( "lobster", "water", PutInPot );
Cook( "chicken", "coconut", BoomBoom );
```

Map-reduce idea: An Example (part 3)

Functions can be anonymous (like classes)

```
Cook( "lobster",
      "water",
      function(x) { alert("pot " + x); }  );
Cook( "chicken",
      "coconut",
      function(x) { alert("boom " + x); } );
```

Map-reduce idea: An Example (part 4)

```
var a = [1,2,3];
for (i=0; i<a.length; i++) {
    a[i] = a[i] * a[i];
}
for (i=0; i<a.length; i++) {
    alert(a[i]);
}
```

Doing something to every element of an array can be done by a function using a function pointer for what needs to be done.

Map-reduce idea: An Example (part 4)

```
var a = [1,2,3];
for (i=0; i<a.length; i++) {
    a[i] = a[i] * a[i];
}
for (i=0; i<a.length; i++) {
    alert(a[i]);
}
```

Doing something to every element of an array can be done by a function using a function pointer for what needs to be done.

```
function map(fn, a)
{
    for (i = 0; i < a.length; i++) {
        a[i] = fn(a[i]);
    }
}

map( function(x){return x*x;}, a );
map( alert, a );
```

Map-reduce idea: An Example (part 5)

```
function sum(a)
{
    var s = 0;
    for (i = 0; i < a.length; i++)
        s += a[i];
    return s;
}

function join(a)
{
    var s = "";
    for (i = 0; i < a.length; i++)
        s += a[i];
    return s;
}

alert(sum([1,2,3]));
alert(join(["a","b","c"]));
```

Map-reduce idea: An Example (part 6)

```
function reduce(fn, a, init)
{
    var s = init;
    for (i = 0; i < a.length; i++)
        s = fn( s, a[i] );
    return s;
}

function sum(a)
{
    return reduce( function(a, b){ return a + b; }, a, 0 );
}

function join(a)
{
    return reduce( function(a, b){ return a + b; }, a, "" );
}
```

Map-reduce idea

- The `map` function is embarrassingly parallel.

Map-reduce idea

- The `map` function is embarrassingly parallel.
- The `reduce` function is nearly embarrassingly parallel.

Map-reduce idea

- The `map` function is embarrassingly parallel.
- The `reduce` function is nearly embarrassingly parallel.
- Someone else can write generic map and reduce code that scales to thousands of systems with massive data sets that span thousands of disks and can tolerate failures of components.... How do we go around doing that?

Map-reduce idea

- The `map` function is embarrassingly parallel.
- The `reduce` function is nearly embarrassingly parallel.
- Someone else can write generic map and reduce code that scales to thousands of systems with massive data sets that span thousands of disks and can tolerate failures of components.... How do we go around doing that?
- Now as long as we can recast our problem as a map-reduce problem, it will automatically scale to thousands of processes!

Map-reduce implementation

- How to scale a parallel program to thousands of processes successfully?

Map-reduce implementation

- How to scale a parallel program to thousands of processes successfully?
- How to deal with massive amounts of data that doesn't fit on one system?

Map-reduce implementation

- How to scale a parallel program to thousands of processes successfully?
- How to deal with massive amounts of data that doesn't fit on one system?
  - Distributed file system.
  - Move code to data instead of the other way around.

Map-reduce implementation

- How to scale a parallel program to thousands of processes successfully?
- How to deal with massive amounts of data that doesn't fit on one system?
  - Distributed file system.
  - Move code to data instead of the other way around.
- How to deal with machines and components failing while the program is running?

MapReduce Paradigm

- MapReduce is a programming model and an associated implementation for processing and generating large data sets.

MapReduce Paradigm

- MapReduce is a programming model and an associated implementation for processing and generating large data sets.
- Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

MapReduce Paradigm

- MapReduce is a programming model and an associated implementation for processing and generating large data sets.
- Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.
- Allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Introduced by Google. Used internally for all major computations on around 1m servers! Amazon leases servers to run map reduce computations (EC2 and S3 programs). Microsoft has developed Dryad (a super set of Map-Reduce).

MapReduce Programming Model (1)

The user writes three methods: the `Map`, `Reduce`, and a main driver that creates the MapReduce specification object.

MapReduce Programming Model (1)

The user writes three methods: the `Map`, `Reduce`, and a main driver that creates the MapReduce specification object.

- `Map`
  - Takes a list of input pairs of *key*s and *value*s, *(K, V)* and produces a set of intermediate key/value pairs.

MapReduce Programming Model (1)

The user writes three methods: the `Map`, `Reduce`, and a main driver that creates the MapReduce specification object.

- `Map`
  - Takes a list of input pairs of *keys* and *values*, (K, V) and produces a set of intermediate key/value pairs.
  - The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the `Reduce` function.

MapReduce Programming Model (1)

The user writes three methods: the `Map`, `Reduce`, and a main driver that creates the MapReduce specification object.

- `Map`
  - Takes a list of input pairs of *key*s and *value*s, (K, V) and produces a set of intermediate key/value pairs.
  - The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the `Reduce` function.
- The `Reduce`
  - Accepts an intermediate key `K` and a set of values for that key.

MapReduce Programming Model (1)

The user writes three methods: the `Map`, `Reduce`, and a main driver that creates the MapReduce specification object.

- `Map`
  - Takes a list of input pairs of *keys* and *values*, (K, V) and produces a set of intermediate key/value pairs.
  - The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the `Reduce` function.
- The `Reduce`
  - Accepts an intermediate key `K` and a set of values for that key.
  - It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation.

MapReduce Programming Model (1)

The user writes three methods: the `Map`, `Reduce`, and a main driver that creates the MapReduce specification object.

- `Map`
  - Takes a list of input pairs of *keys* and *values*, *(K, V)* and produces a set of intermediate key/value pairs.
  - The MapReduce library groups together all intermediate values associated with the same intermediate key K and passes them to the `Reduce` function.
- The `Reduce`
  - Accepts an intermediate key `K` and a set of values for that key.
  - It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per Reduce invocation.
  - The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

MapReduce Programming Model (2)

- MapReduce Specification Object.
  - Contains names of input/output files and optional tuning parameters.

MapReduce Programming Model (2)

- MapReduce Specification Object.
  - Contains names of input/output files and optional tuning parameters.
  - The user then invokes the MapReduce function, passing it the specification object.

MapReduce Programming Model (2)

- MapReduce Specification Object.
  - Contains names of input/output files and optional tuning parameters.
  - The user then invokes the MapReduce function, passing it the specification object.
  - The user's code is linked together with a MapReduce library.

MapReduce Programming Model (2)

- MapReduce Specification Object.
  - Contains names of input/output files and optional tuning parameters.
  - The user then invokes the MapReduce function, passing it the specification object.
  - The user's code is linked together with a MapReduce library.
  - When the code runs, it leverages the MapReduce implementation to deal with distributed file systems, distributed computing, fault tolerance, reporting etc. For example, *Hadoop* provides a MapReduce framework.

MapReduce Programming Model (2)

- MapReduce Specification Object.
  - Contains names of input/output files and optional tuning parameters.
  - The user then invokes the MapReduce function, passing it the specification object.
  - The user's code is linked together with a MapReduce library.
  - When the code runs, it leverages the MapReduce implementation to deal with distributed file systems, distributed computing, fault tolerance, reporting etc. For example, *Hadoop* provides a MapReduce framework.
  - The output will be distributed in multiple files across the cluster, just as the input is. If the output is small, then it can be collected together, if not, we have to leave it distributed.

A MapReduce Pseudo-code Example

Consider the problem of counting the number of occurrences of each word in a large collection of documents.

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

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

MapReduce Exercise

- **Case Analysis or Capitalization Probability**: In a collection of text documents, find the percentage capitalization for each letter of the alphabet.

MapReduce Examples (1)

- Distributed Grep: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.

MapReduce Examples (1)

- Distributed Grep: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- Count of URL Access Frequency: The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.

MapReduce Examples (1)

- Distributed Grep: The map function emits a line if it matches a supplied pattern. The reduce function is an identity function that just copies the supplied intermediate data to the output.
- Count of URL Access Frequency: The map function processes logs of web page requests and outputs <URL, 1>. The reduce function adds together all values for the same URL and emits a <URL, total count> pair.
- Distributed Sort: The map function extracts the key from each record, and emits a <key, record> pair. The reduce function emits all pairs unchanged. This computation depends on the partitioning and ordering facilities that are provided in a MapReduce implementation.

MapReduce Examples (2)

- Reverse Web-Link Graph:
  - The map function outputs <target, source> pairs for each link to a target URL found in a page named source.
  - The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: <target, list(source)>

MapReduce Examples (3)

- Inverted Index:
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.

MapReduce Examples (3)

- Inverted Index:
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.
  - The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.

MapReduce Examples (3)

- Inverted Index:
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.
  - The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.
  - The set of all output pairs forms a simple inverted index.

MapReduce Examples (3)

- Inverted Index:
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.
  - The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.
  - The set of all output pairs forms a simple inverted index.
  - It is easy to augment this computation to keep track of word positions.

MapReduce Examples (3)

- Inverted Index:
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.
  - The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.
  - The set of all output pairs forms a simple inverted index.
  - It is easy to augment this computation to keep track of word positions.
  - How about computing a relevance of each document?

MapReduce Examples (3)

- Inverted Index:
  - The map function parses each document, and emits a sequence of <word, document ID> pairs.
  - The reduce function accepts all pairs for a given word, sorts the corresponding document IDs and emits a <word, list(document ID)> pair.
  - The set of all output pairs forms a simple inverted index.
  - It is easy to augment this computation to keep track of word positions.
  - How about computing a relevance of each document?
  - How about also keeping track of the context (most relevant lines of text where it was found)

MapReduce Exercise

- Find the number of citations for each patent in a patent reference data set. The format of the input is

  citing_patent, cited_patent

  Describe a MapReduce algorithm to solve this problem.

MapReduce Exercise

- Find the number of citations for each patent in a patent reference data set. The format of the input is

  citing_patent, cited_patent

  Describe a MapReduce algorithm to solve this problem.
- Find the top N most frequently cited patents. The format of the input is

  citing_patent, cited_patent

  Describe a MapReduce algorithm to solve this problem.
  Hint: This will take two passes.

References

- *MapReduce: Simplified Data Processing on Large Clusters* by Jeffrey Dean and Sanjay Ghemawat, Google Inc. Appeared in: OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
- *Can Your Programming Language Do This?* by Joel Spolsky. http://www.joelonsoftware.com/items/2006/08/01.html