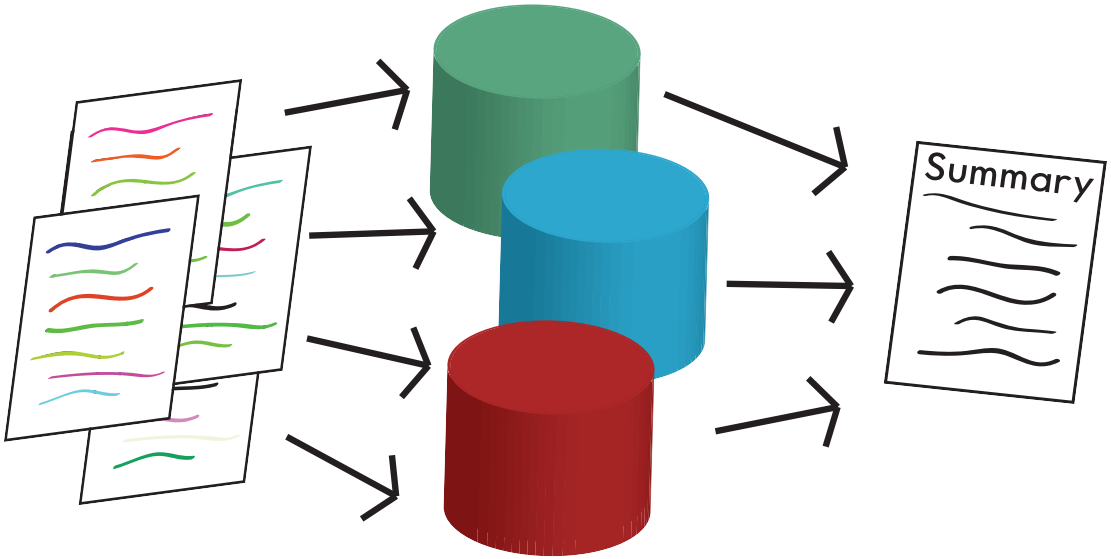
Abstractions for Data Intensive Computing

Severin Hacker, Robert J. Simmons, and Carsten Varming

Carnegie Mellon University

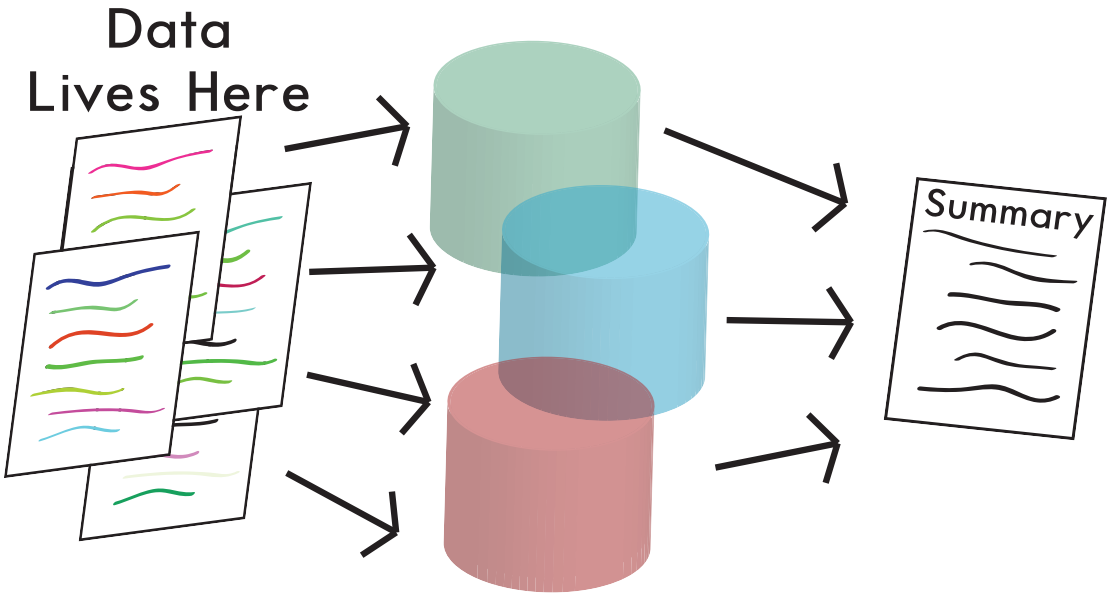
December 19, 2007

Carnegie Mellon



Anatomy of a data-intensive computing problem

Carnegie Mellon



Hadoop and MapReduce are more “File-centric”

Carnegie Mellon

Netezza is more “Database-centric”

Carnegie Mellon

Netezza is a “Data Warehouse Appliance”

- FPGA + minimal computer right next to the disk
- Filter queries as fast as the disk spins

Netezza does not provide a clean abstraction

- Interacting C++ and SQL code
- Awkward C++ style
- Is SQL the right way to query?
- MapReduce seems like an abstraction for this architecture. . .

Hypothesis:

- There are interesting problems at the intersection of what Netezza and MapReduce do well
- By specifying this intersection, we can "Write once, Run ~~Anywhere~~ on Netezza and Yahoo Hadoop"
- Something of the flavor of Sawzall?

Carnegie Mellon

Carnegie Mellon

No proper “map” in Netezza

- Can’t count all occurrences of every word
- Concentrate on “reduce”
- No sorting

What is the closest we can get to “map”?

- Old: zero or more (key, value) pairs
- New: zero or one (key, value) pairs
- Weak form of MapReduce: ModReduce

ModReduce is a SQL-esque variant of MapReduce

Input: "mod" = `mod(x)`, "reduce" = `red(y)`

Output: `select red(mod_val(x)) from myTable where mod_sel(x) group by mod_key(x)`

Problems:

- What if `mod(x)` is expensive? Recompute?
- The output of `red(x)` is only a single value! Top 10 query?

Carnegie Mellon

Input:

`select f(x) from myTable where g(y) group by h(z)`

Output:

- If `g(z)` holds, "mod" emits key $=$ `h(y)`, val $=$ `x`
- "reduce" is just `f`

Problems:

- Real SQL has table joins...

Proof of concept implementation: SRC Code

- Make programming the Netezza less painful
- Allow code to run on Netezza (yes) and Hadoop (not yet)

Features:

- Strongly typed w/ type inference
- Models SQL's `null` values
- Use external functions from C++, Java

Querying with SRC Code:

```
=> SELECT count(email) FROM spamdata WHERE
substr("get rich",msg) GROUP BY email

...response...

=> MODREDUCE contains("get rich",email,msg),
count FROM spamdata

...response...
```

Implementing functions with SRC Code:

```
external substr : (string,string) -> bool;

contains(pattern,key,contents) =
  if substr(pattern,contents)
  then some (key,1)
  else null;

add(a,b) = a + b;

export count {merge=add, unit=0};
```

In comparison, Sawzall has . . .

- . . . a more domain-specific notion of aggregators-as-tables
- . . . a similar notion of operating over pre-structured records
- . . . an interpreter (no compilation)

- Netezza and Hadoop
  - Focused and super fast vs. general and super scalable
  - Program both/either with SQL/ModReduce/MapReduce
  - Least-common-denominator approach isn't perfect
- Compilers are good!
  - Sawzall can't implement aggregators...
- Portability and good abstraction
  - Thinking across platforms is a good approach
  - ModReduce is not the last word!