Feliss: Flexible distributed computing framework with light-weight checkpointing

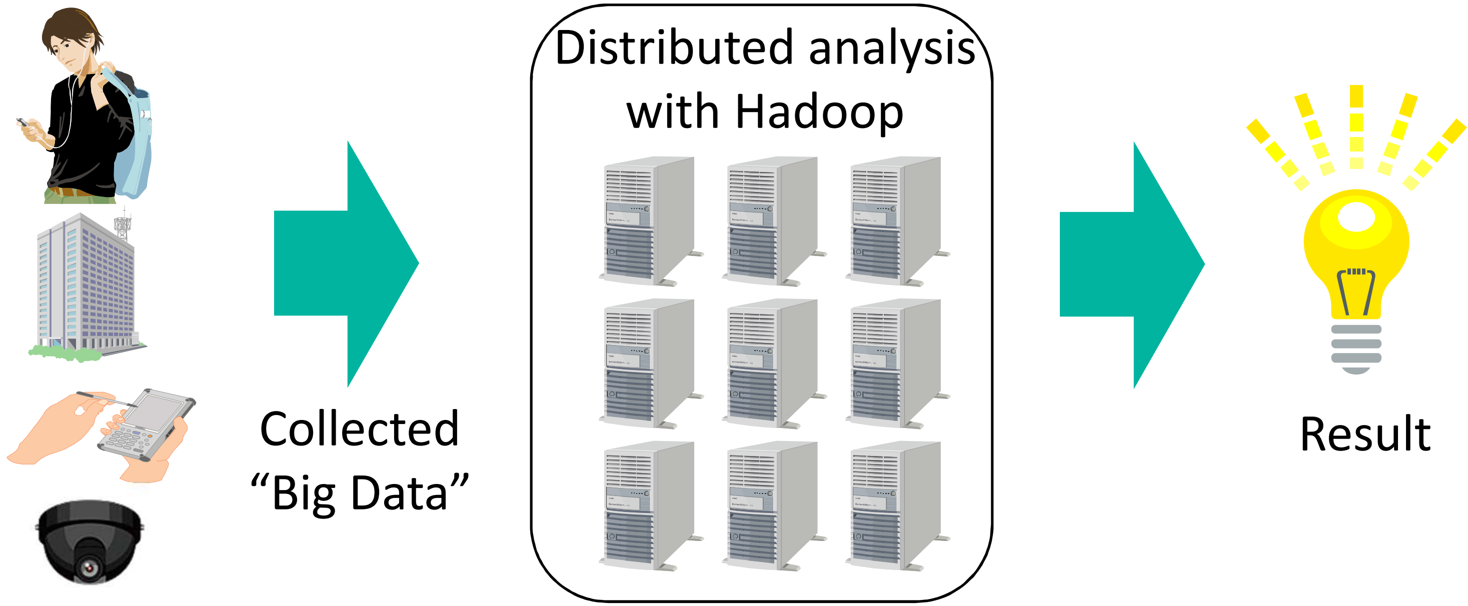
Takuya Araki, Kazuyo Narita and Hiroshi Tamano

NEC Corporation

Background

Big Data analysis is becoming common

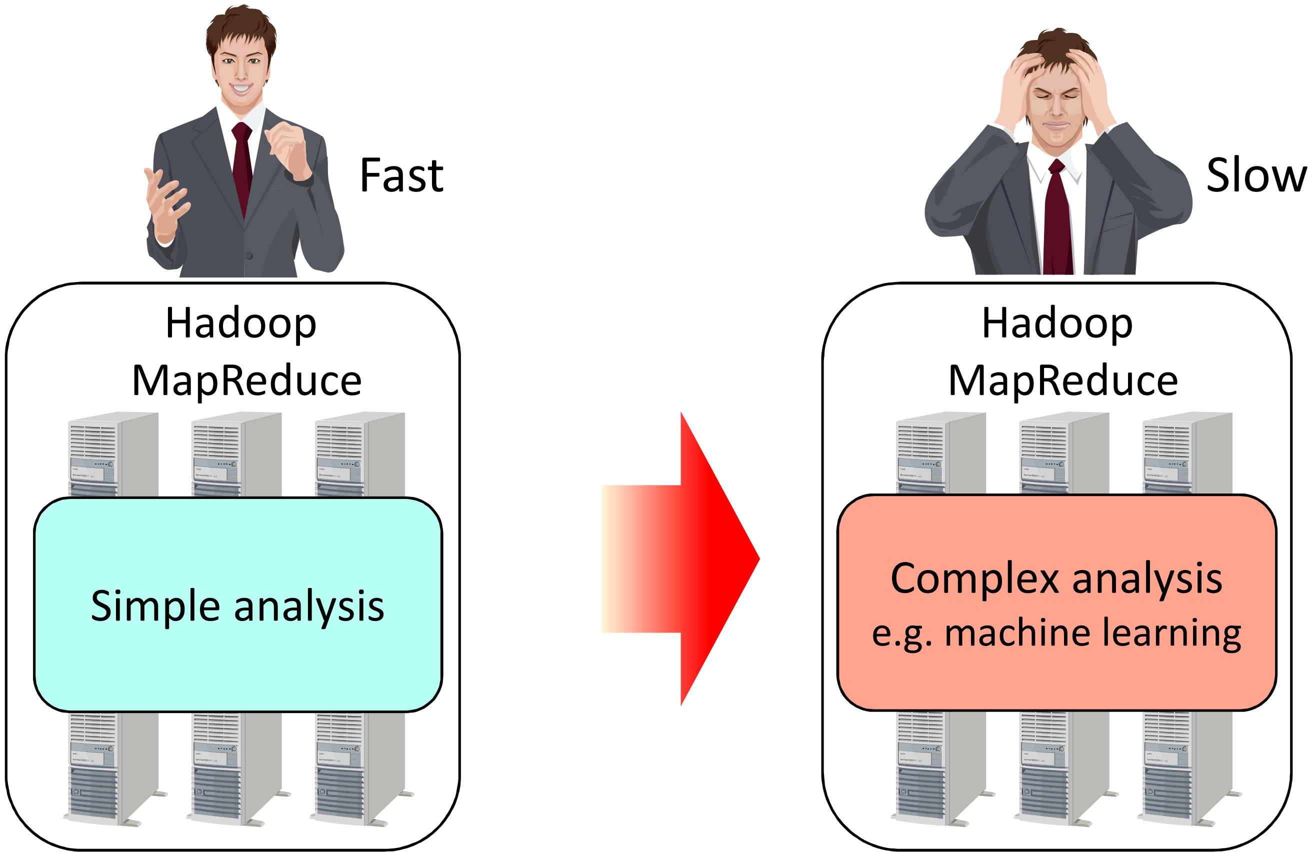
- sensors, the Web, business transactions, etc.
- Hadoop / MapReduce is commonly used



Motivation

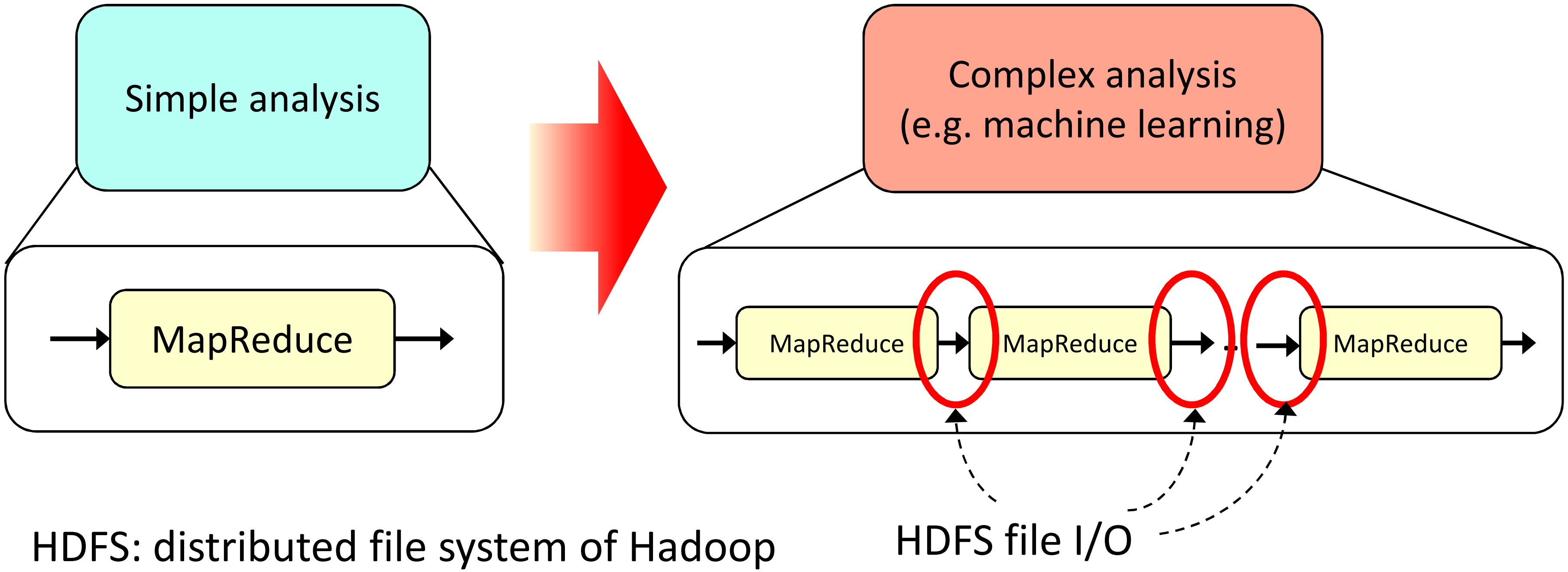
Hadoop / MapReduce is:

- efficient if it fits well with the problem
- not efficient, otherwise



Why Hadoop / MapReduce is slow?

- Many MapReduces need to be combined to implement complex analysis
  - e.g. iterative computation
- They need to communicate through HDFS file I/O, which causes large overhead

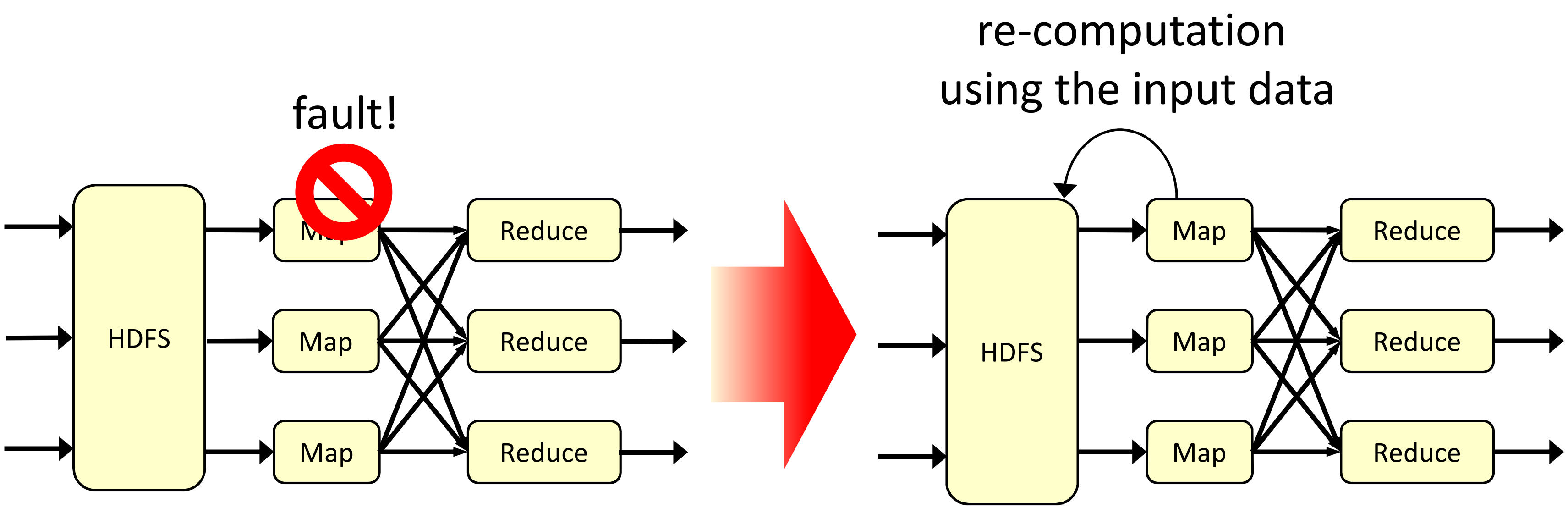


HDFS: distributed file system of Hadoop

Why HDFS file I/O is needed?

For fault-tolerance!

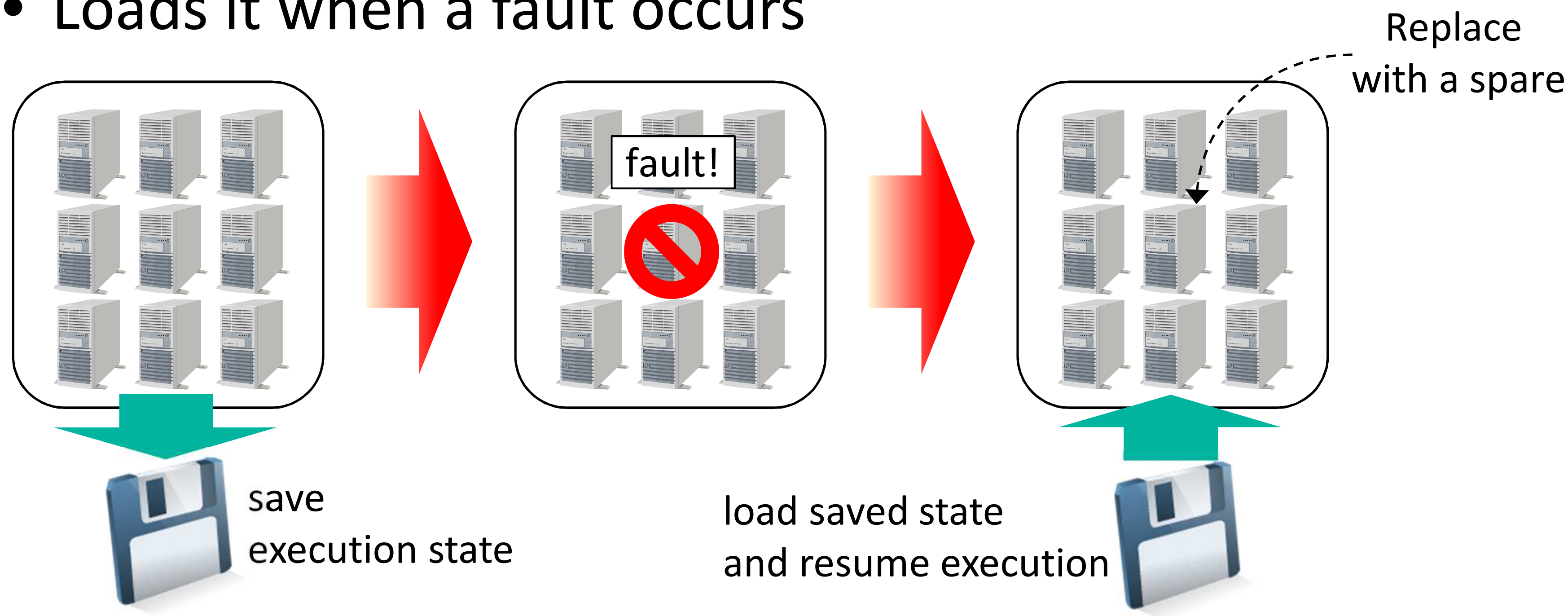
- If a fault occurs, the system re-computes lost data from the input data on HDFS
  - input data need to be on "stable storage"



Any other fault-tolerance methods?

Checkpointing!

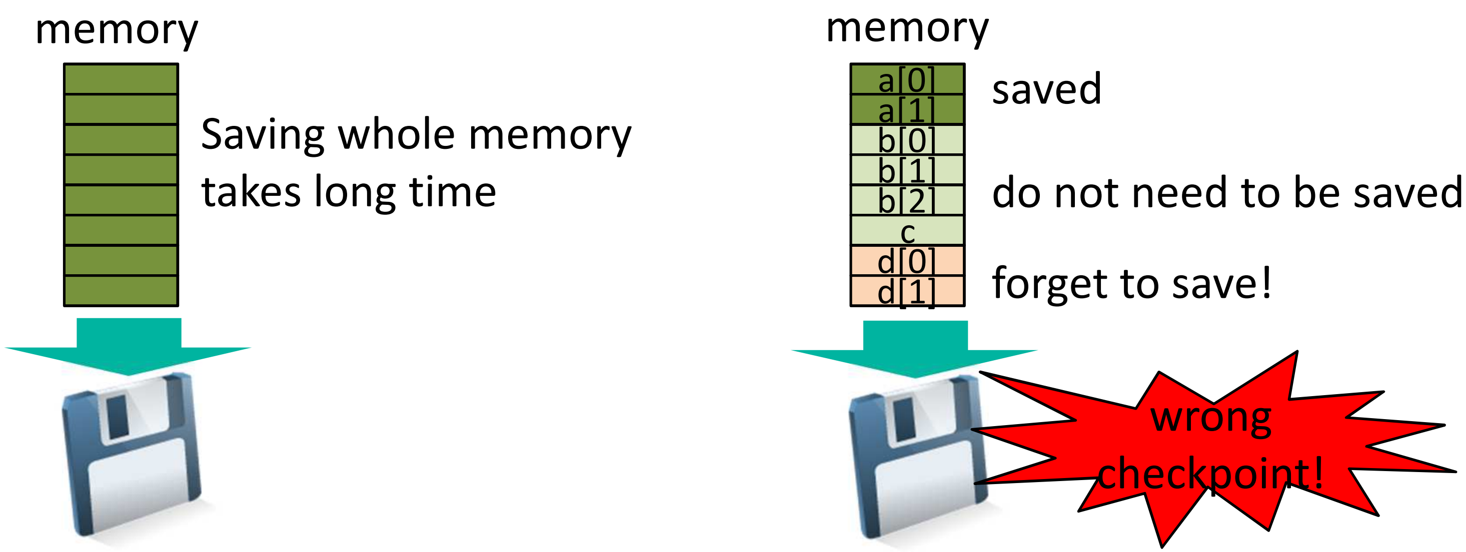
- Saves execution state periodically
- Loads it when a fault occurs



No need to use HDFS for communication between MapReduces

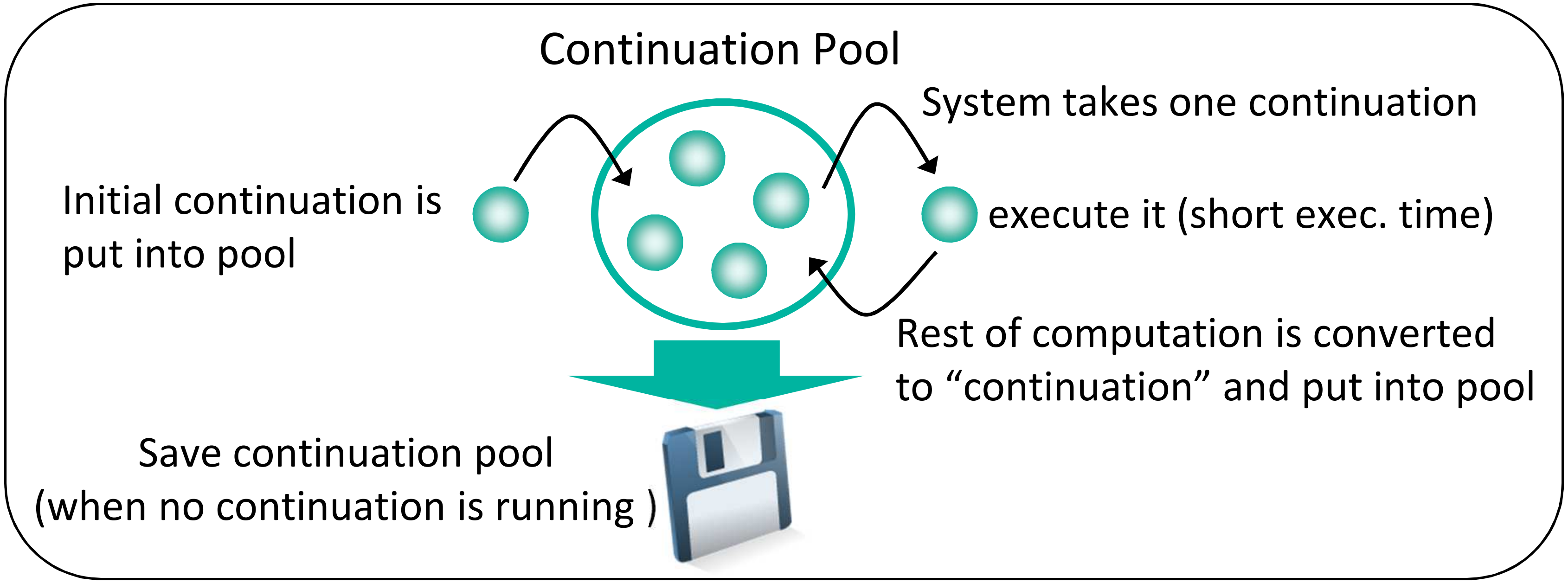
What is the problem of checkpointing?

- Saving entire memory
  - Takes long time:
    - 10GB memory & 100MB/s disk bandwidth: 100 seconds
- Saving variables specified by a programmer
  - Error prone:
    - If he/she forgets to save a variable, it does not work



Our proposal: continuation-based checkpointing

Programs are written as follows: (continuation is similar to “task”)



By saving the continuation pool, whole execution state can be saved, because it contains “rest of computation” as continuations

Only necessary memory is saved, without specifying variables explicitly

Rest of this talk

Implementation of continuation-based checkpointing

Feliss: distributed computing framework
with continuation-based checkpointing

Distributed checkpointing

Improved MapReduce and MPI

Evaluation and related work

Implementation of continuation-based checkpointing

Feliss: distributed computing framework
with continuation-based checkpointing

Distributed checkpointing

Improved MapReduce and MPI

Evaluation and related work

Implementation of continuation-based checkpointing (1/2)

Rest of the computation is converted to continuation

- Example:

Convert function call g

```
void sample(int a)  {
  f();
  g(a);
}
```

```
void sample(int a)  {
  f();
  put_pool (make_cont(g,a));
}
```

put the created continuation into the pool

create continuation of function g with arg a

Only small number of conversion is needed

- continuation exec. time only affects checkpoint interval time; granularity of continuation need not be too small

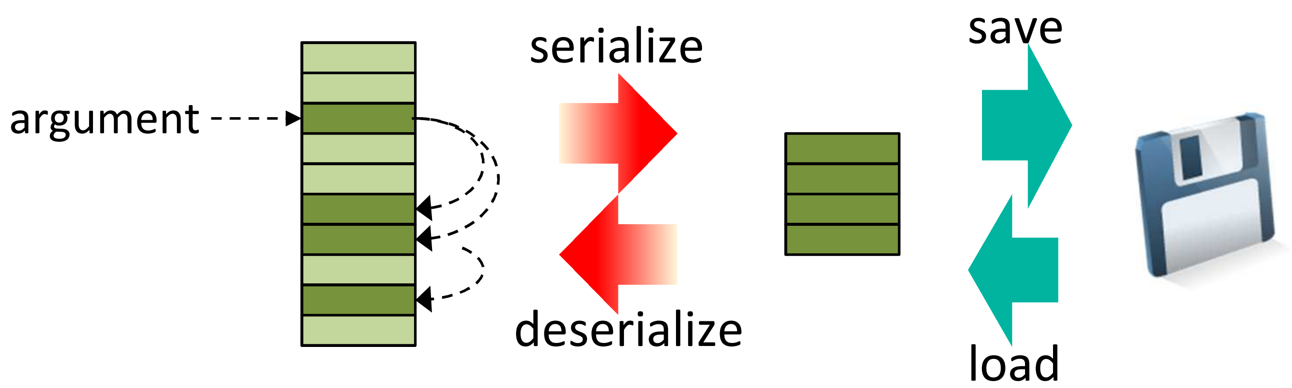
When MapReduce is used, this is hidden by MapReduce layer

Implementation of continuation-based checkpointing (2/2)

- In C++ on Linux
- Data structure of continuation is simple:

pointer to function	argument 1	argument 2	argument 3	...

- Continuations are saved after "serialization"
  - Serialization converts data with pointers into contiguous data
  - Boost::serialization is used for serializing arguments



  - "Symbol name" is used for serializing pointer to function
    - using dladdr / dlsym (provided by Linux)

Implementation of continuation-based checkpointing

Feliss: distributed computing framework with continuation-based checkpointing

Distributed checkpointing

Improved MapReduce and MPI

Evaluation and related work

Feliss: distributed computing framework with continuation-based checkpointing

- Implemented distributed computing framework using continuation-based checkpointing

- Features:
  - distributed checkpointing
  - RPC (remote procedure call)
  - non-blocking (callback based) synchronization
  - Improved MapReduce
  - MPI (for supporting matrix operations)

- Explain these 3 features

Implementation of continuation-based checkpointing

Feliss: distributed computing framework
with continuation-based checkpointing

Distributed checkpointing

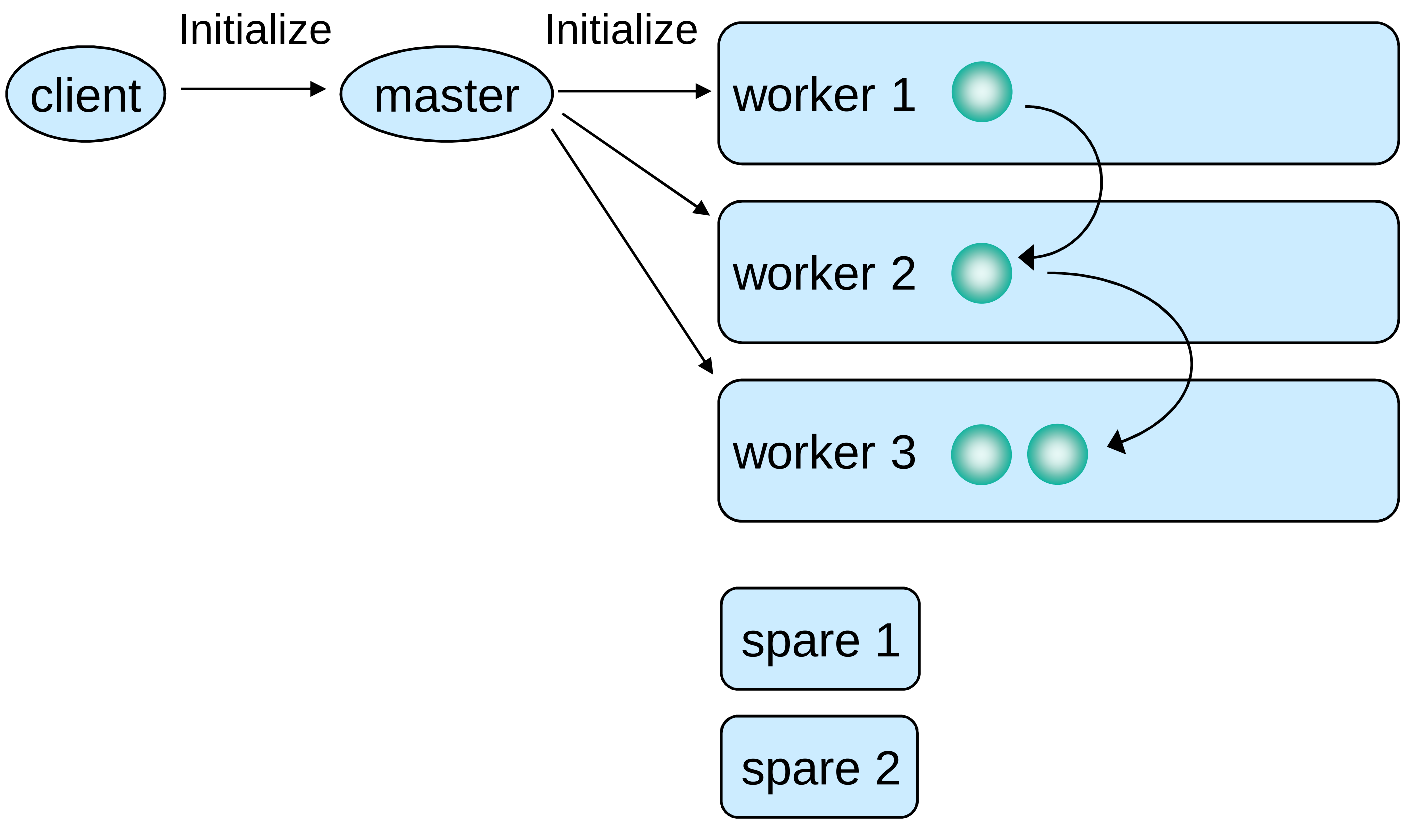
Improved MapReduce and MPI

Evaluation and related work

Distributed checkpointing (1/3)

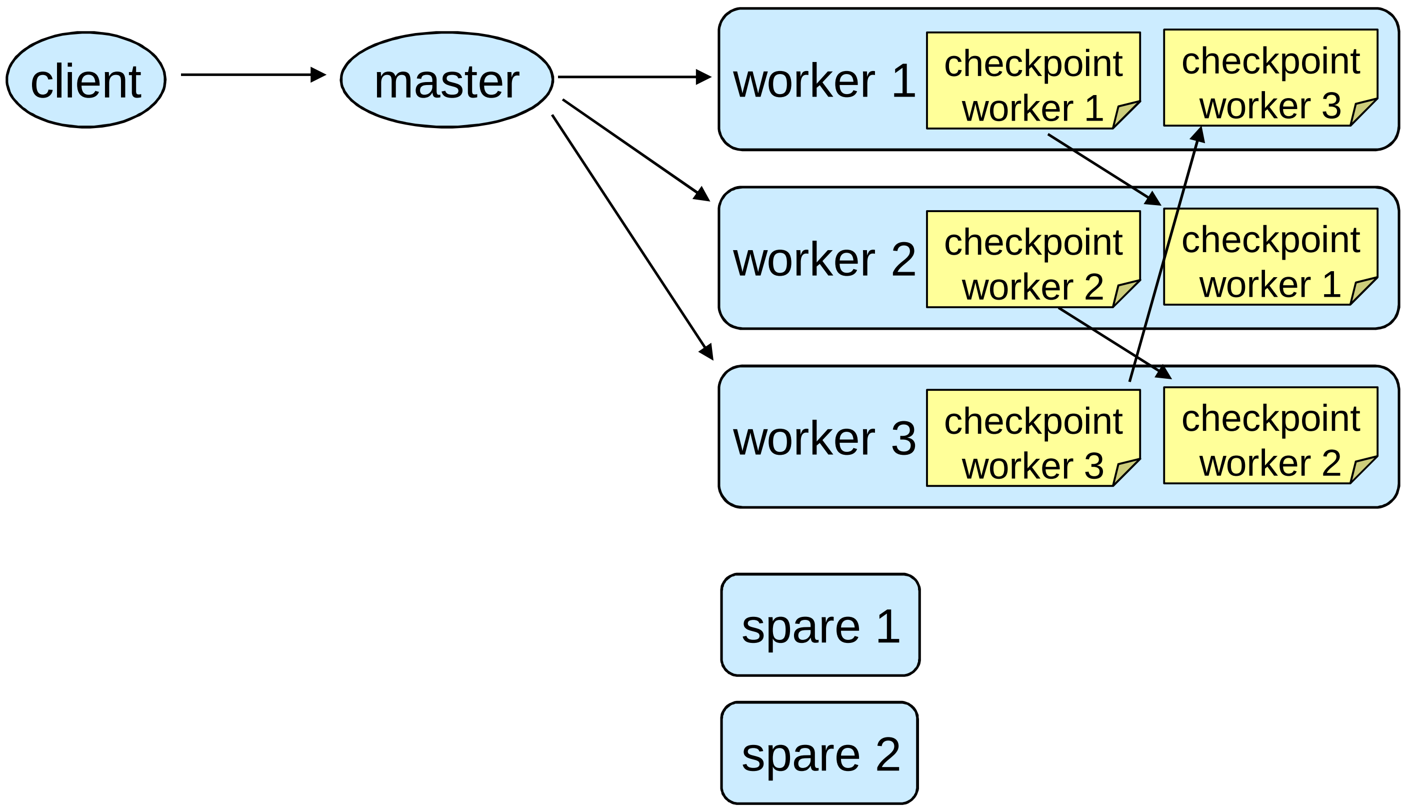
Three kinds of servers:
- master, worker, spare

Workers do actual distributed computation
- by sending and receiving continuations using RPC



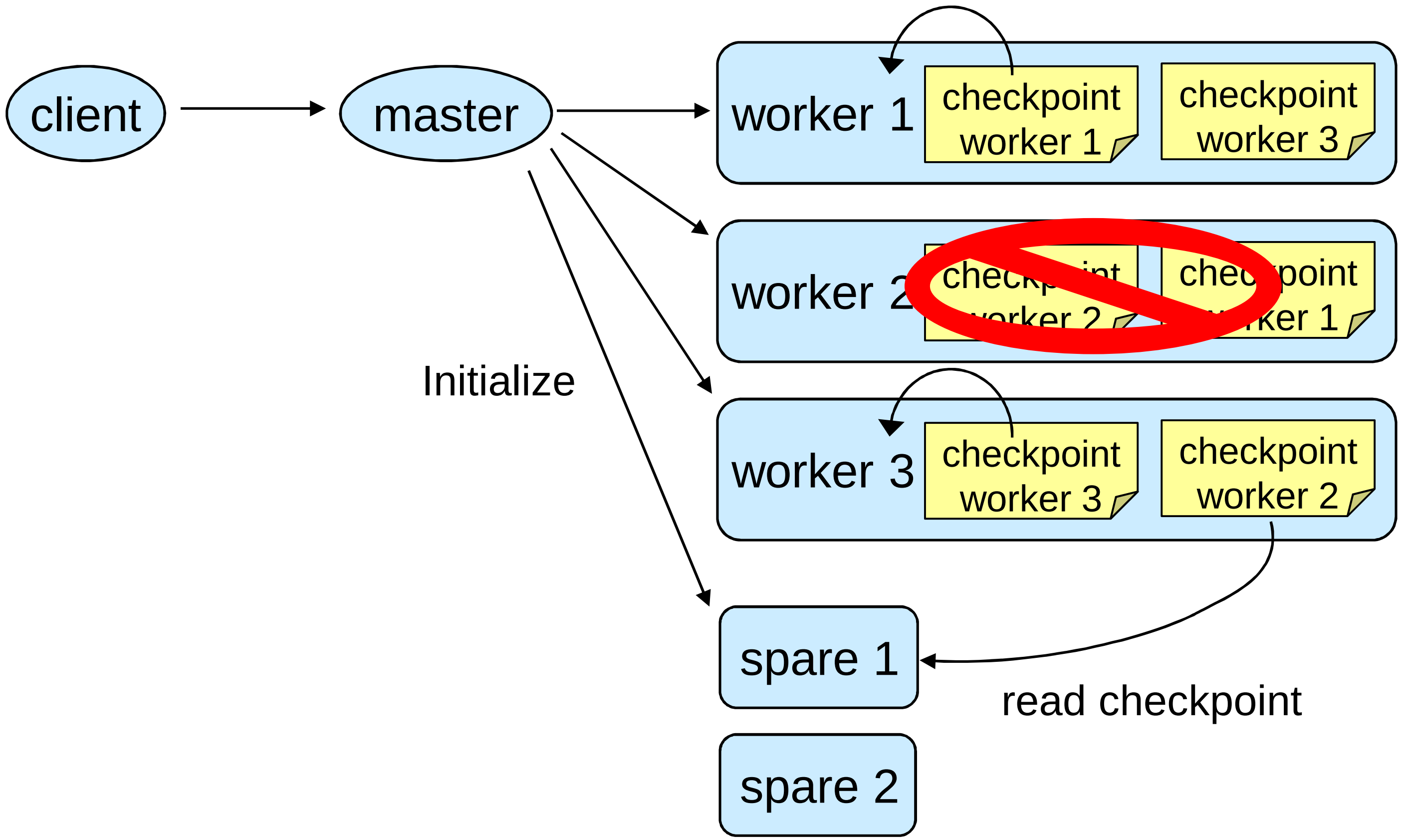
Distributed checkpointing (2/3)

Master orders workers to take checkpoints periodically
Checkpoints are sent to other workers for preparing fault



Distributed checkpointing (3/3)

Master checks if workers are running correctly
When a fault occurs, it restarts workers from checkpoint
- if the worker is still working, a local checkpoint is used
- otherwise, a spare reads a checkpoint from a worker that has it



Implementation of continuation-based checkpointing

Feliss: distributed computing framework with continuation-based checkpointing

Distributed checkpointing

Improved MapReduce and MPI

Evaluation and related work

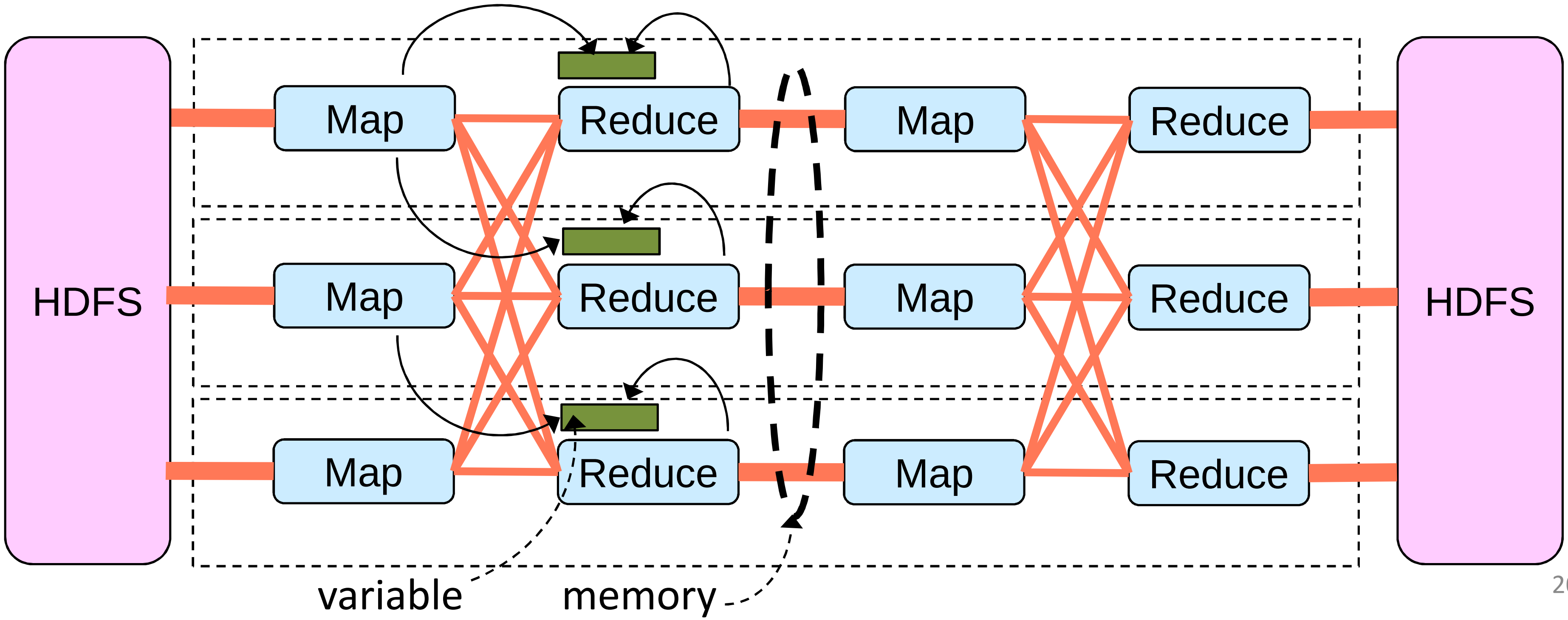
Improved MapReduce

Similar to original MapReduce

- map, reduce, partition and combine functions

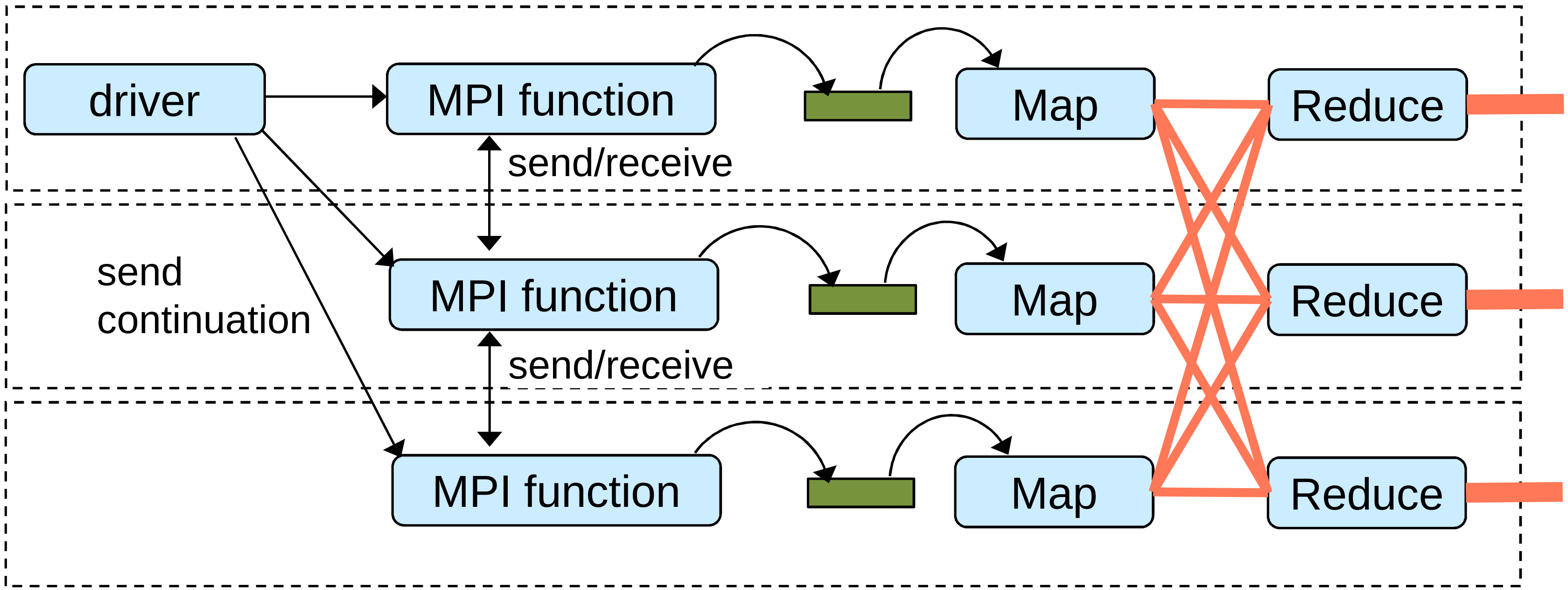
Difference:

- MapReduces can be connected through memory
- these functions can access to arbitrary distributed variables, which leads to more flexible computation



MPI (Message Passing Interface)

- With MPI, matrix operations can be efficiently written, which are commonly used in machine learning algorithms
- Continuation of the top level MPI function is sent to all the workers to realize SPMD
  - the function is same as the usual MPI program
- Can be used with MapReduce
  - communicate through variables



Implementation of continuation-based checkpointing

Feliss: distributed computing framework
with continuation-based checkpointing

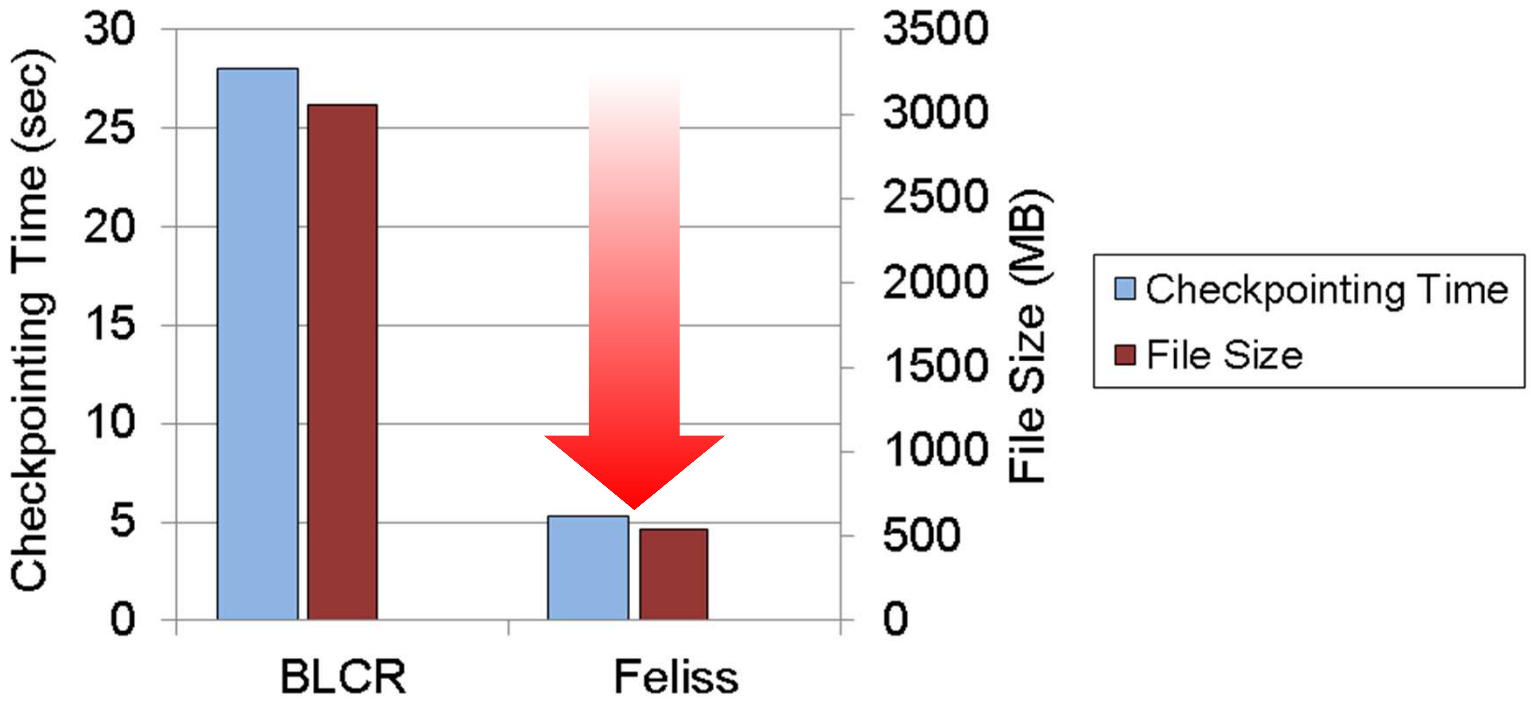
Distributed checkpointing

Improved MapReduce and MPI

Evaluation and related work

Evaluation (1/2)

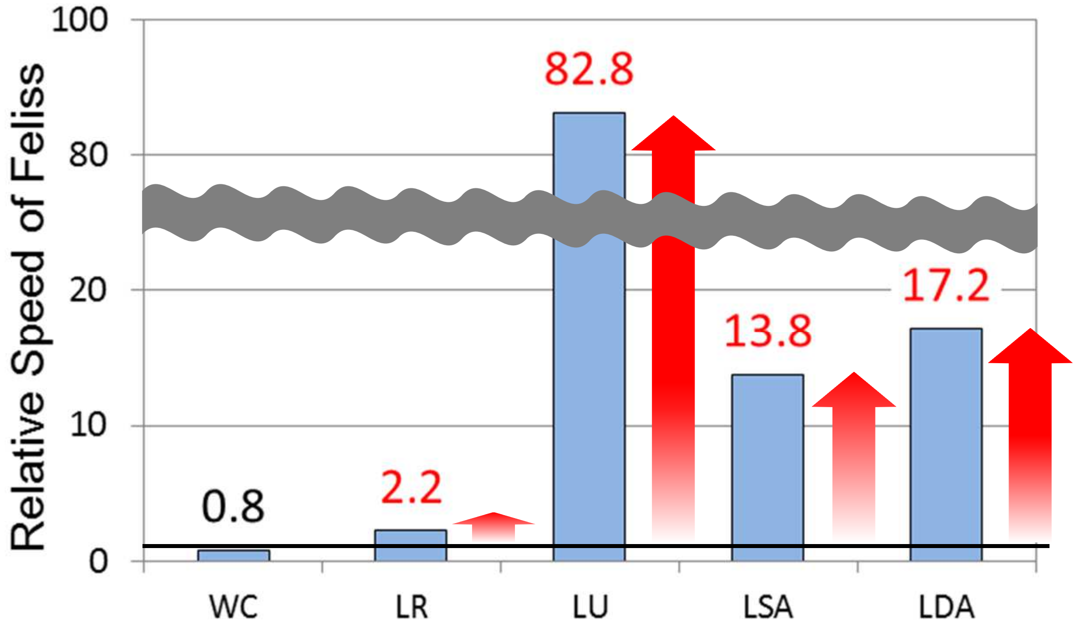
- Compared checkpointing time and size with BLCR, which saves entire memory image (one server)
- The program holds std::map with 20 million data
- 5.3 times faster and 5.6 times smaller than BLCR



Evaluation (2/2)

Compared speed of Feliss with other frameworks (18 servers/72 CPU)

Application	Input	used functionality	compared with
Word Count (WC)	English Wikipedia	MapReduce	Hadoop
Logistic Regression (LR)	20GB of vector	RPC	Spark
LU decomposition (LU)	4320 x 4320	MPI	
Latent semantic analysis (LSA)	English Wikipedia	MapReduce + MPI	Mahout (Hadoop)
Latent Dirichlet allocation (LDA)	English Wikipedia (1/32)	Multiple MapReduces	



Other than WC, Feliss is much faster than other frameworks

Related work

- Checkpointing
  - BLCR, Libckpt, Condor, etc. save entire memory
  - Method proposed by Cores et al. saves only live variables with the help of a compiler
    - Our method does not require special compiler

- Distributed computing framework
  - Haloop, Twister, and Spark only support limited computation patterns and do not support matrix operation
  - Piccolo and Distributed GraphLab only support limited data structures like table or graph

Conclusion

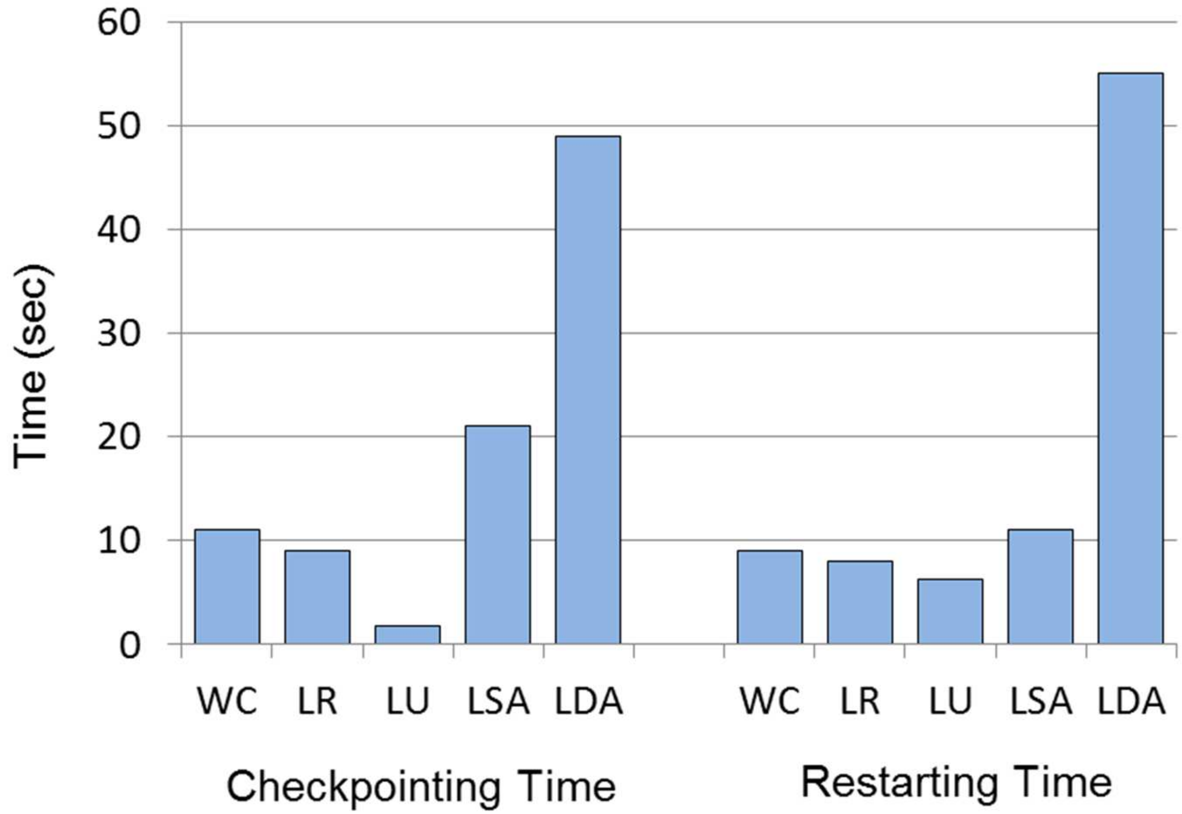
- Proposed continuation-based checkpointing
  - only saves necessary memory without specifying variables explicitly
- Implemented distributed computing framework called Feliss using it
  - confirmed that Feliss is much faster than existing frameworks
- Future work
  - improve checkpointing performance
    - supporting asynchronous / incremental checkpointing
  - utilize checkpointing for resource management
    - e.g. migrating process to less loaded servers

Backup

Performance of checkpointing and restarting

- Checkpointing time: time to take one checkpoint
- Restarting time: time to restart from failure

Both are short enough with these applications



Scalability

- WC, LR and LDA shows good scalability
- Scalability of LU and LSA was lower, because they hit the hardware limit of network bandwidth
  - would be improved with better network like 10G Ethernet or InfiniBand, instead of Gigabit Ethernet