PHÂN TÍCH DỮ LIỆU LỚN

Giảng viên: Nguyễn Tu Trung, Trần Mạnh Tuấn
BM HTTT, Khoa CNTT, Trường ĐH Thủy Lợi

Hà Nội, 2019

Nội dung

- Mô hình MapReduce
- Môi trường lập trình Hadoop MapReduce
- Lập trình bài toán WordCount

Mô hình MapReduce

- Lịch sử ra đời MapReduce
- MapReduce là gì?
- Quản lý thực thi công việc
- Thực hiện công việc trên MapReduce
- Ví dụ: Bài toán đếm từ
- Hàm map, reduce
- Ưu điểm của mô hình MapReduce
- Kiến trúc các thành phần
- Cơ chế hoạt động
- Ứng dụng của MapReduce

Lịch sử ra đời MapReduce

- Trước khi Google công bố mô hình MapReduce
  - Bùng nổ của dữ liệu (hàng petrabyte)
  - Nhu cầu thực hiện xử lý các nghiệp vụ trên lượng dữ liệu khổng lồ là thách thức lớn lúc bấy giờ
  - Doanh nghiệp đang gặp vấn đề tương tự khi muốn tìm một giải pháp tốn ít chi phí và hiệu năng thể hiện cao
- Trong khi nghiên cứu, một nhóm nhân viên của Google đã khám phá ra một ý tưởng để giải quyết nhu cầu xử lý lượng dữ liệu lớn là việc cần phải có hệ thống nhiều các máy tính và cần có các thao tác để xử lý đồng bộ trên hệ thống đó
- Nhóm nghiên cứu đã xác định được 2 thao tác cơ bản là Map và Reduce, nó được lấy cảm hứng từ phong cách lập trình hàm (Functional Programming)

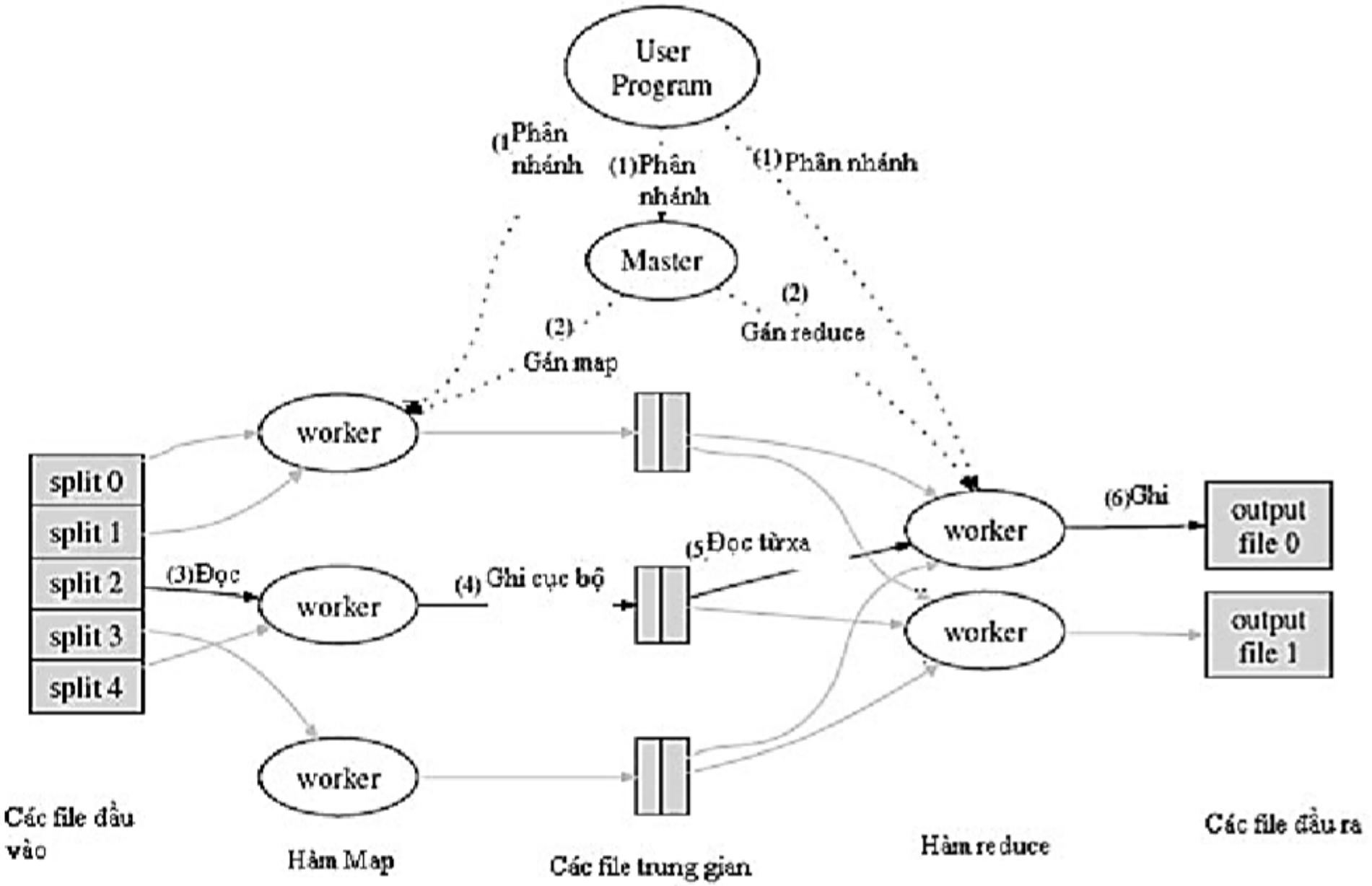
MapReduce là gì?

- Với ý tưởng trên, Google phát triển thành công mô hình MapReduce:
  - Là mô hình dùng cho xử lý tính toán song song và phân tán trên hệ thống phân tán
  - B1: Phân rã từ nghiệp vụ chính (do người dùng muốn thể hiện) thành các công việc con để chia từng công việc con này về các máy tính trong hệ thống thực hiện xử lý một cách song song
  - B2: Thu thập lại các kết quả
- Theo tài liệu "MapReduce: Simplified Data Processing on Large Clusters" của Google: "MapReduce là mô hình lập trình và thực thi song song các xử lý và phát sinh các tập dữ liệu lớn"
- Với mô hình này, các doanh nghiệp đã cải thiện được đáng kể về hiệu suất xử lý tính toán trên dữ liệu lớn, chi phí đầu tư rẻ và độ an toàn cao

Quản lý thực thi công việc

- Hệ thống định nghĩa:
  - Một máy trong hệ thống đóng vai trò là master
  - Các máy còn lại đóng vai trò các worker (dựa trên kiến trúc Master-Slave)
- Master chịu trách nhiệm quản lý toàn bộ quá trình thực thi công việc trên hệ thống như
  - Tiếp nhận công việc
  - Phân rã công việc thành công việc con
  - Phân công các công việc con cho các worker
- Worker chỉ làm nhiệm vụ thực hiện công việc con được giao (thực hiện hàm map hoặc hàm reduce)

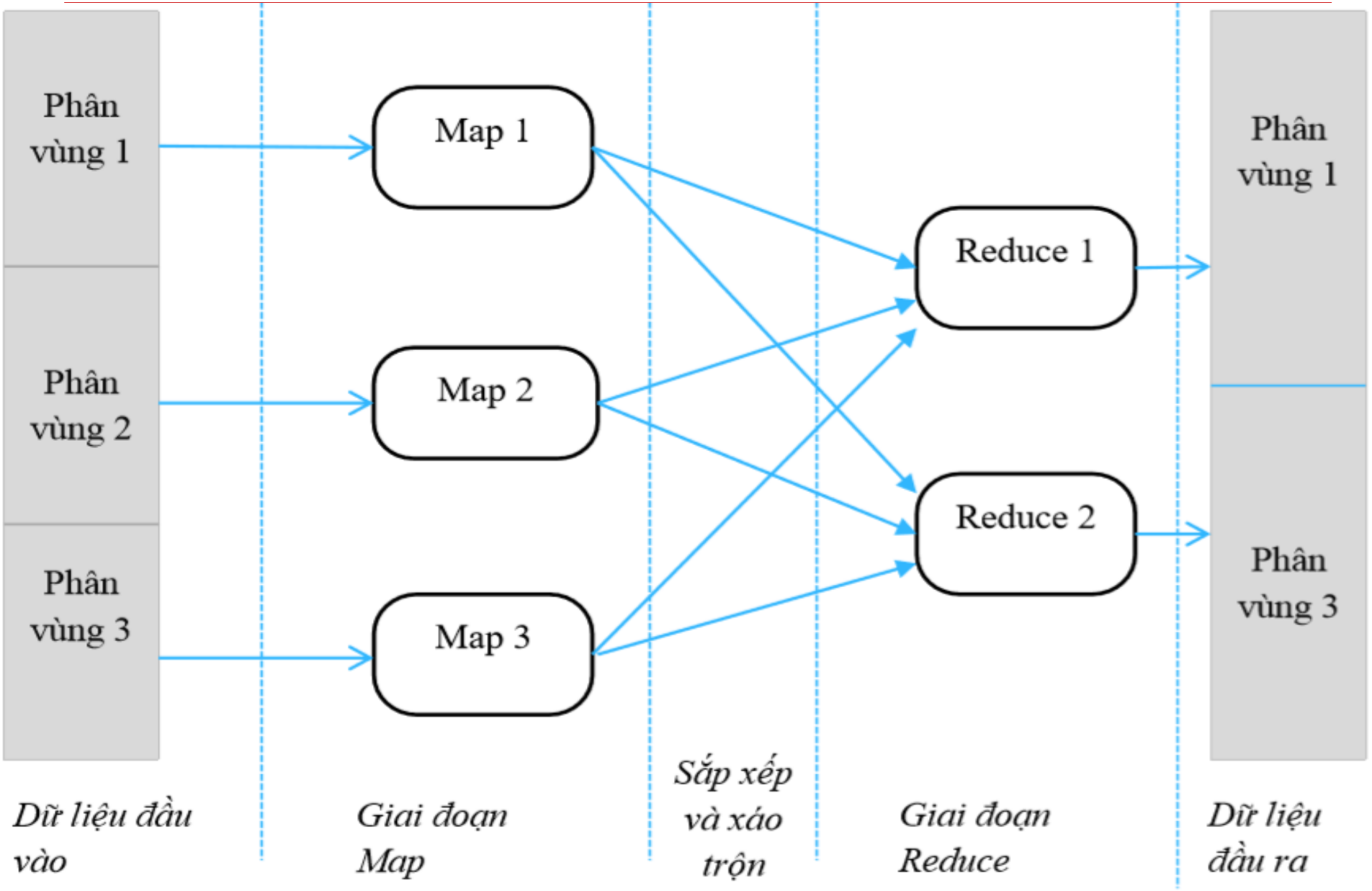
Quản lý thực thi công việc



Thực hiện công việc trên MapReduce

- Quy trình thực hiện công việc
  - B1: Chia dữ liệu đầu vào thành các mảnh dữ liệu
  - B2: Thực hiện công việc **Map** trên từng mảnh dữ liệu đầu vào
    - => Xử lý song song các mảnh dữ liệu trên nhiều máy tính trong cụm
  - B3: Tổng hợp kết quả trung gian (Sắp xếp, trộn)
  - B4: Sau khi tất cả công việc Map hoàn thành, thực hiện công việc **Reduce** trên từng mảnh dữ liệu trung gian
    - => Thực hiện song song các mảnh dữ liệu trung gian trên nhiều máy tính trong cụm
  - B5: Tổng hợp kết quả hàm Reduce để cho kết quả cuối cùng

Thực hiện công việc trên MapReduce



Ví dụ: Bài toán đếm từ

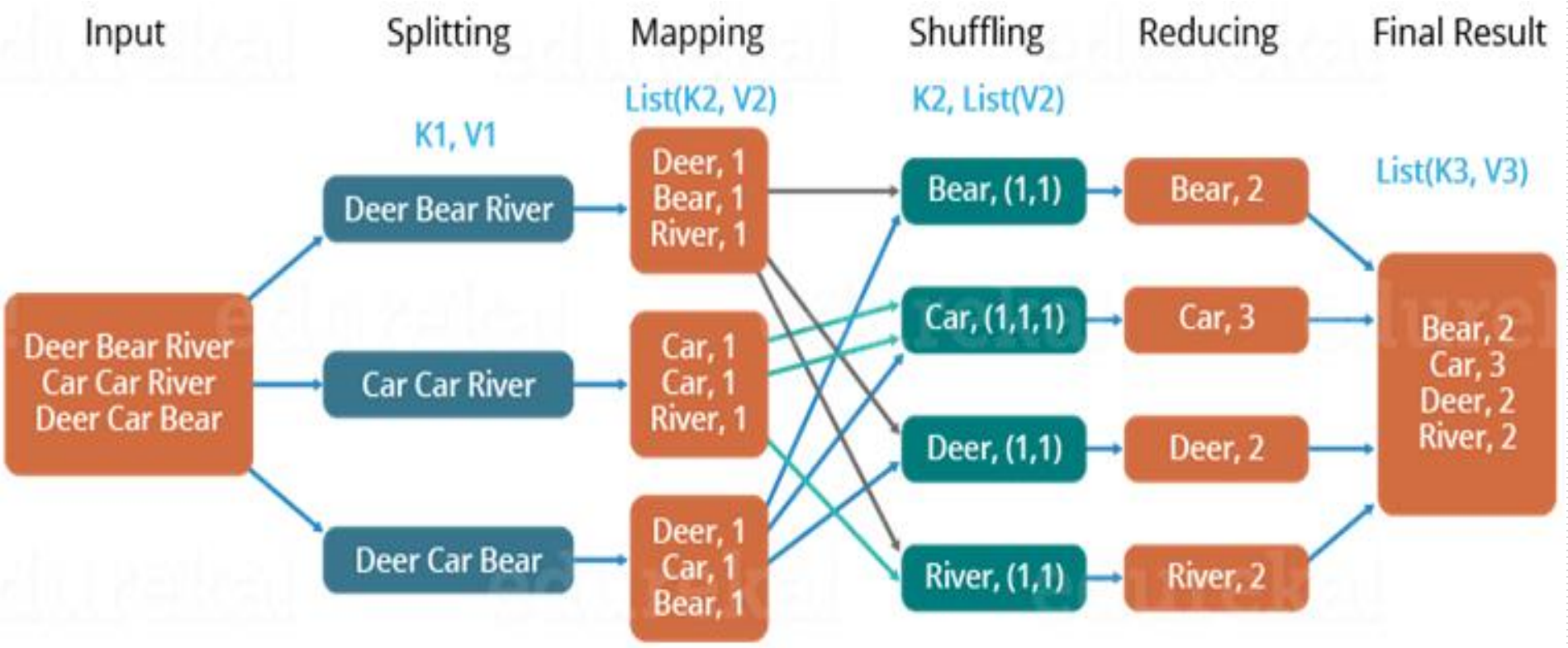
- File text example.txt có nội dung như sau:
  - Dear, Bear, River, Car, Car, River, Deer, Car, Bear
- Nhiệm vụ: đếm tần xuất các từ trong example.txt sử dụng MapReduce
- Ý tưởng: tìm các từ duy nhất và đếm số lần xuất hiện của các từ này
- Quy trình thực hiện:
  - B1-Splitting: Giả sử chia đầu vào 3 phần
    - P1: Dear, Bear, River
    - P2: Car, Car, River
    - P3: Deer, Car, Bear
  - B2-Mapping: Với mỗi phần, duyệt từng từ, gán giá trị 1 cho mỗi từ (lý do: ko tính lặp lại, mỗi từ xuất hiện 1 lần) để thu được list (từ, 1)
    - Dear, Bear, River -> (Dear,1), (Bear,1), (River,1)
    - Car, Car, River -> (Car,1), (Car,1), (River,1)
    - Dear, Car, Bear - > (Dear,1), (Car,1), (Bear,1)

Ví dụ: Bài toán đếm từ

- Quy trình thực hiện (tiếp):
  - B3: Shorting & Shuffling: Nhóm các giá trị cùng một từ, kết quả thu được
    - Bear, (1,1); Car, (1,1,1); Dear, (1,1); River, (1,1)
  - B4: Reducing: Tính tổng các giá trị cùng một từ
    - Bear, (1,1) -> (Bear, 2)
    - Car, (1,1,1) -> (Car,3)
    - Dear, (1,1) -> (Dear,2)
    - River, (1,1) -> (River,2)
  - B5: Tổng hợp kết quả Reduce được kết quả cuối cùng
    - (Bear, 2); (Car,3); (Dear,2); (River,2)

Ví dụ: Bài toán đếm từ

- Quy trình đếm từ dựa trên mô hình MapReduce



Hàm map, reduce

- MapReduce dùng hai thao tác chính cho việc thực thi công việc là hàm Map và hàm Reduce
  - Hàm Map tiếp nhận mảnh dữ liệu input, rút trích thông tin cần thiết các từng phần tử (ví dụ: lọc dữ liệu, hoặc trích dữ liệu) tạo kết quả trung gian
  - Hệ thống thực hiện một bước trung gian để trộn và sắp xếp lại kết quả
  - Hàm Reduce tổng hợp kết quả trung gian, tính toán để cho kết quả cuối cùng
- Hàm Map và Reduce được xem là phần xử lý quan trọng nhất trong mô hình MapReduce, do người dùng định nghĩa tùy theo nhu cầu sử dụng
- Giai đoạn reduce chỉ bắt đầu khi giai đoạn map kết thúc

Hàm map, reduce

- MapRedue định nghĩa dữ liệu (cấu trúc và không cấu trúc) dưới dạng cặp khóa/giá trị (key/value)
  - Ví dụ: key có thể là tên của tập tin (file) và value nội dung của tập tin, hoặc key là địa chỉ URL và value là nội dung của URL…
  - Việc định nghĩa dữ liệu thành cặp key/value này linh hoạt hơn các bảng dữ liệu quan hệ 2 chiều truyền thống (khóa chính – khóa ngoại)
- Hàm map và reduce làm việc với khối dữ liệu dạng này
- Hàm map:
  - Input: một cặp (keyIn, valIn)
  - Output: danh sách các cặp (keyInt, valInt) trung gian (Intermediate)
  - Biểu diễn hình thức: map (keyIn, valIn) -> list (keyInt, valInt)
- Hàm reduce:
  - Input: một cặp (keyInt, list(valInt))
  - Output: danh sách các cặp (keyOut, valOut)
  - Biểu diễn hình thức: reduce (keyInt, list(valInt)) -> list (keyOut, valOut)

Ưu điểm của mô hình MapReduce

- Xây dựng từ mô hình lập trình hàm và lập trình song song
- Giúp cải thiện tốc độ tính toán trên tập dữ liệu lớn bằng cách tăng tốc độ đọc ghi và xử lý dữ liệu
- Có thể áp dụng hiệu quả có nhiều bán toán.
- Ẩn đi các chi tiết cài đặt và quản lý như:
  - Quản lý tiến trình song song và phân tán
  - Quản lý, sắp xếp lịch trình truy xuất I/O
  - Theo dõi trạng thái dữ liệu
  - Quản lý số lượng lớn dữ liệu có quan hệ phụ thuộc nhau
  - Xử lý lỗi
  - Cung cấp mô hình lập trình đơn giản => Người dùng quan tâm chủ yếu đến hàm map, reduce
  - …

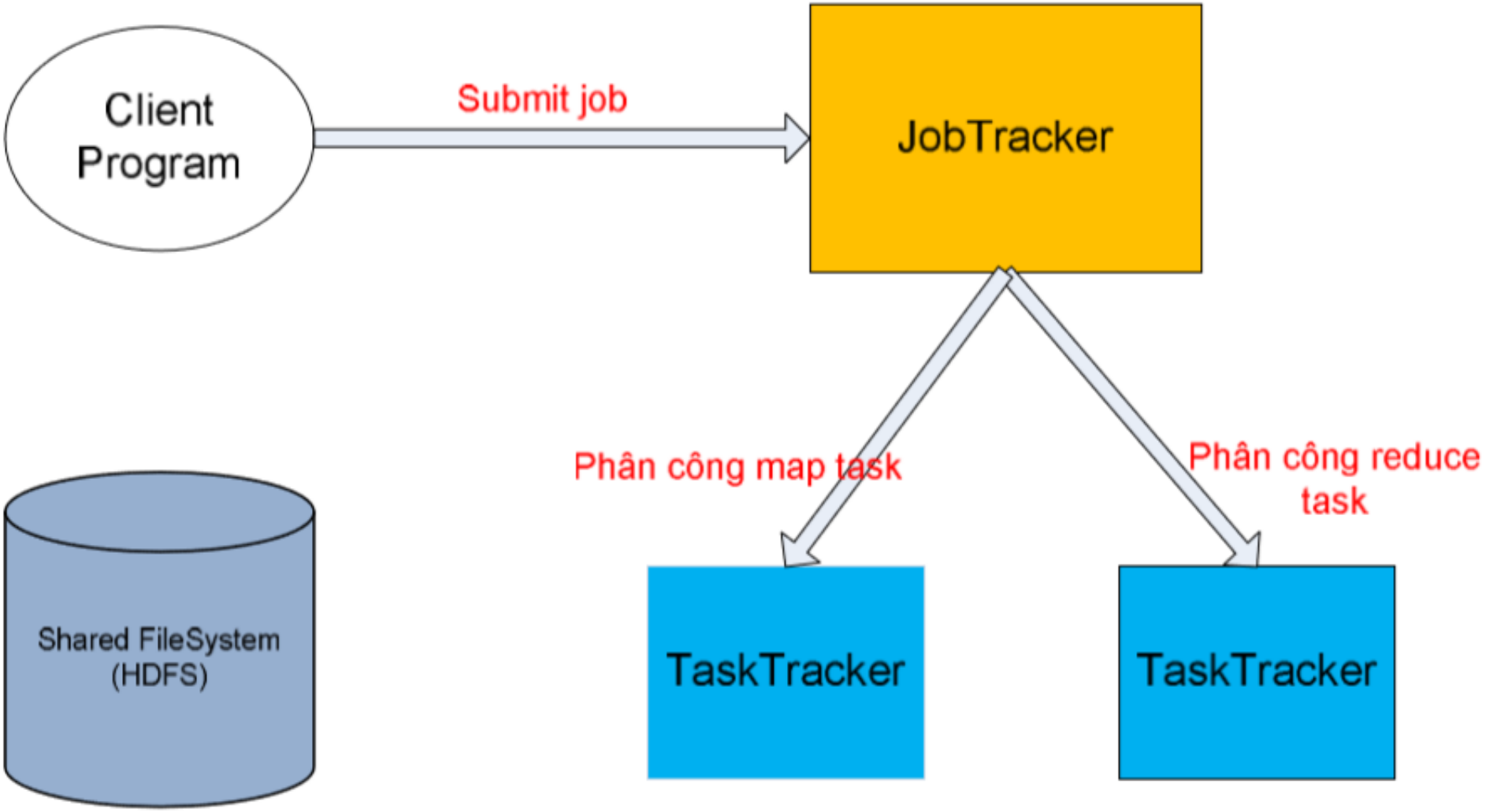
Kiến trúc các thành phần

- Xét một cách trừu tượng, Hadoop MapReduce gồm 4 thành phần chính riêng biệt
  - Client Program: Chương trình HadoopMapReduce client sử dụng để chạy một MapReduce Job
  - JobTracker:
    - Tiếp nhận job và đảm nhận vai trò điều phối job này
    - Có vai trò như bộ não của Hadoop MapReduce
    - Chia nhỏ job thành các task
    - Lập lịch phân công các task (map task, reduce task) này đến các tasktracker để thực hiện
    - Có cấu trúc dữ liệu riêng để sử dụng cho mục đích lưu trữ: lưu lại tiến độ tổng thể của từng job, lưu lại trang thái của các TaskTracker để thuận tiện cho thao tác lên lịch phân công task, lưu lại địa chỉ lưu trữ của các output của các TaskTracker thực hiện maptask trả về

Kiến trúc các thành phần

- TaskTracker:
  - Tiếp nhận maptask hay reducetask từ JobTracker để sau đó thực hiện
  - Để giữ liên lạc với JobTracker, Hadoop Mapreduce cung cấp cơ chế gửi heartbeat từ TaskTracker đến JobTracker cho các nhu cầu như thông báo tiến độ của task do TaskTracker đó thực hiện, thông báo trạng thái hiện hành của nó (idle, in-progress, completed)
- HDFS:
  - Hệ thống file phân tán được dùng cho việc chia sẻ các file dùng trong cả quá trình xử lý một job giữa các thành phần trên với nhau

Kiến trúc các thành phần



Cơ chế hoạt động

- Submit Job và chuẩn bị dữ liệu
- Quản lý Job và chia task
- Liên lạc giữa JobTracker và TaskTracker
- Làm việc ở MapTaskTracker
- Làm việc ở ReduceTaskTracker

Submit Job và chuẩn bị dữ liệu

- Client gửi yêu cầu thực hiện job và kèm theo dữ liệu input tới JobTracker
- JobTracker thông báo cho client tình trạng tiếp nhận job
- Nếu tình trạng tiếp nhận hợp lệ, client tiến hành phân rã input này thành các split và ghi vào HDFS
- Client gửi thông báo sẵn sàng để JobTracker biết việc chuẩn bị dữ liệu đã thành công và tiến hành thực hiện job

Quản lý Job và chia Task

- Khi nhận được thông báo sẵn sàng từ client, JobTracker đưa job này vào một danh sách lưu các Job mà các client yêu cầu thực hiện
- Tại một thời điểm JobTracker chỉ thực hiện một job
- Sau khi một job hoàn thành hay bị block, JobTracker sẽ lấy job khác trong list này (FIFO) ra thực hiện
- JobTracker có một job scheduler với nhiệm vụ lấy vị trí các split (từ HDFS do chương trình client tạo) và sau đó tạo một danh sách các task để thực thi
- Mỗi split có một maptask để thực thi => số lượng maptask bằng với số lượng split
- JobTracker lưu trữ thông tin trạng thái và tiến độ của tất cả các task

Liên lạc giữa JobTracker và TaskTracker

- Hadoop cung cấp cho các TaskTracker cơ chế gửi heartbeat đến JobTracker theo chu kỳ thời gian
- Thông tin bên trong heartbeat này cho phép JobTrack biết được TaskTracker này có thể thực thi task hay không
  - Tasktracker có thể cùng lúc chạy nhiều map task và reduce task một cách đồng bộ
  - Số lượng tối đa các task chạy cùng lúc dựa trên số lượng core, số lượng bộ nhớ Ram và kích thước heap (bộ nhớ cần để thực thi một maptask) bên trong TaskTracker
- 2 loại TaskTracker: TaskTracker thực thi maptask, TaskTracker thực thi reduce task

Làm việc ở MapTaskTracker

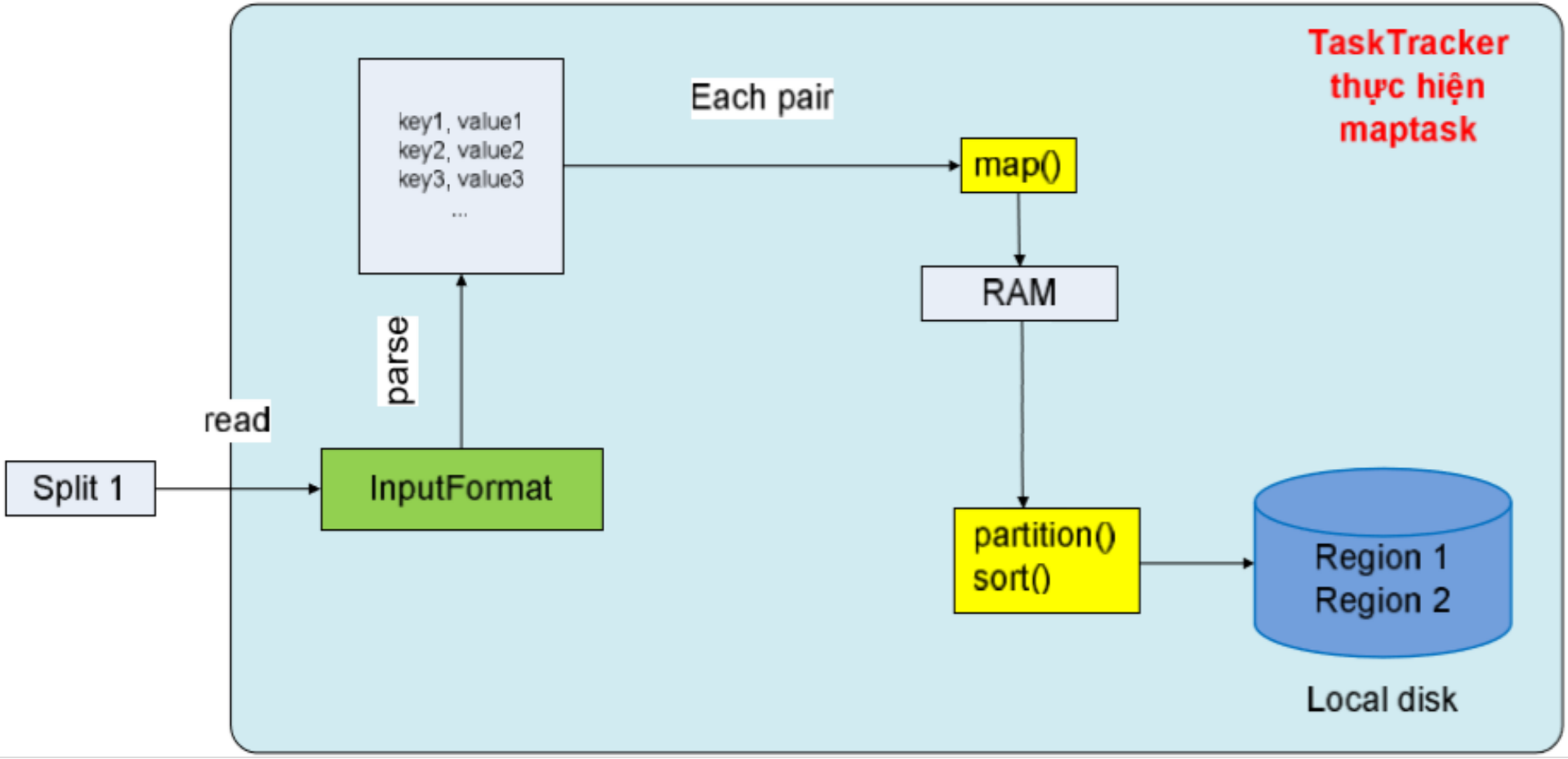
- TaskTracker nhận thực thi maptask, kèm theo là vị trí của input split trên HDFS
- MapTaskTracker nạp dữ liệu của split từ HDFS vào bộ nhớ
- Dựa vào kiểu format của dữ liệu input do client chọn, MapTaskTracker phân tích split này để phát sinh ra tập các record
  - Record này có 2 trường: key và value
  - Ví dụ: với kiểu input format là text, record ứng với mỗi dòng: key là offset đầu tiên của dòng (offset toàn cục), value là chuỗi ký tự của dòng

Làm việc ở MapTaskTracker

- Với tập các record này, MapTaskTracker chạy vòng lặp để lấy từng record làm input cho hàm **map** để trả về output là dữ liệu gồm key và value trung gian (intermediate)
- Dữ liệu output của hàm map được chia thành các mảnh dữ liệu trung gian (dựa trên hàm partition)
- Mỗi mảnh dữ liệu trung gian được sắp sếp theo key trung gian (dựa trên hàm sort) và lưu thành các vùng tương ứng (partition) trên đĩa cục bộ của MapTaskTracker
- Cuối cùng, MapTaskTracker sẽ gửi trạng thái completed của maptask và danh sách các vị trí của các partition (region) output trên localdisk của nó đến JobTracker
- Lưu ý: Từng partition này sẽ ứng với dữ liệu input của reduce task sau này

Làm việc ở MapTaskTracker

- MapTaskTracker thực hiện maptask



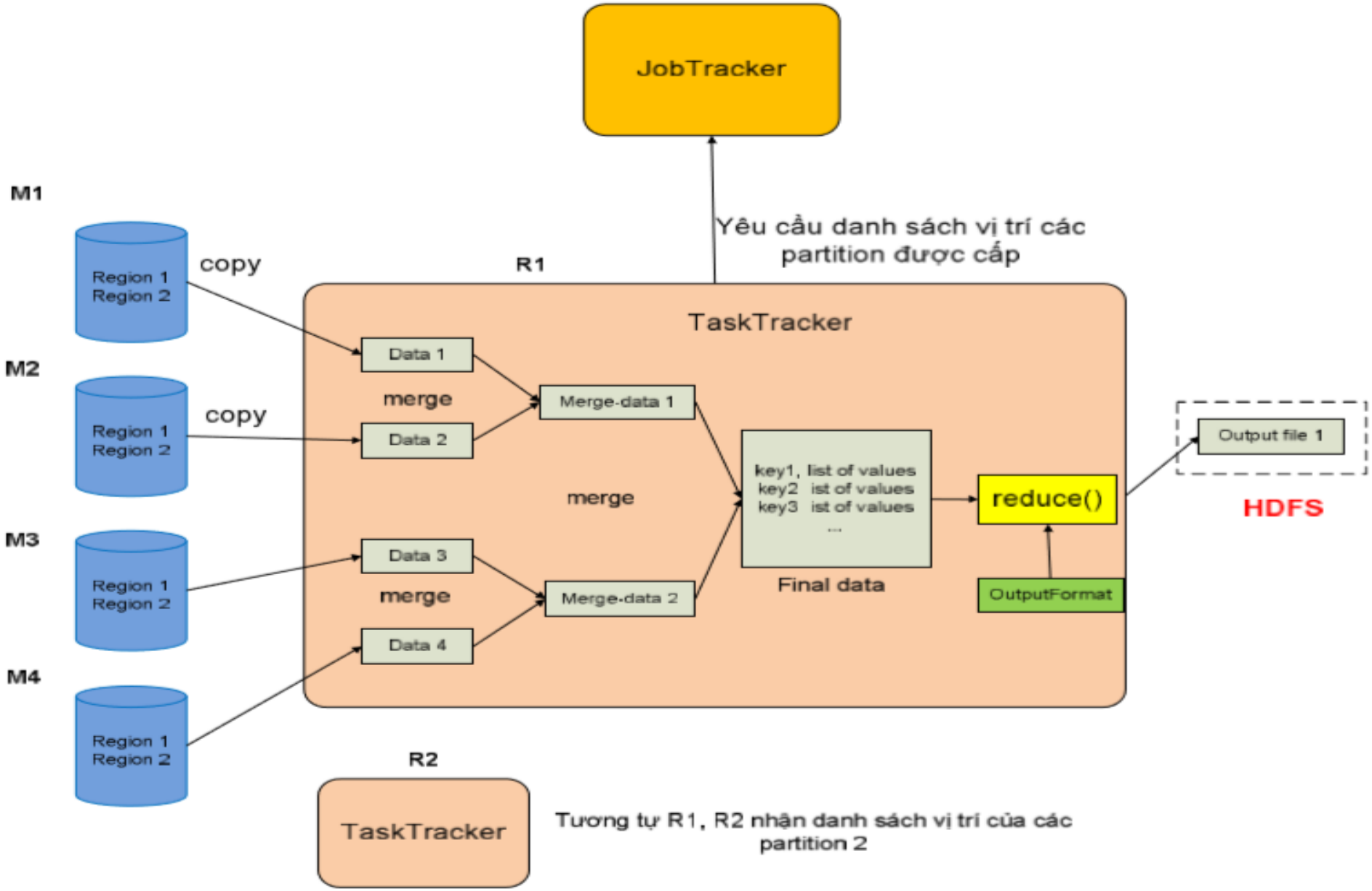
Làm việc ở ReduceTaskTracker

- Input của một ReduceTaskTracker là một region cụ thể cùng danh sách các vị trí lưu dữ liệu của region này trên localdisk của các MapTaskTracker
- Sau khi MapTaskTracker cuối cùng hoàn thành nhiệm vụ, ReduceTaskTracker sẽ nhận được đầy đủ thông tin các vị trí lưu dữ liệu của region mà JobTracker giao cho
- Với danh sách vị trí này, ReduceTaskTracker nạp (copy) dữ liệu từ các vị trí này và thực hiện việc hợp dữ liệu theo key trung gian
  - Quá trình nạp và hợp dữ liệu có thể chạy song song để tăng hiệu suất làm việc
  - Kết quả thu được danh sách các cặp (keyInt,list(valInt))
- TaskTracker chạy vòng lặp để lấy từng record ra làm input cho hàm reduce

Làm việc ở ReduceTaskTracker

- Hàm reduce dựa vào kiểu format của output để thực hiện và trả ra kết quả output thích hợp
- Tất cả các dữ liệu output này sẽ được lưu vào một file và file này sau đó sẽ được ghi xuống HDFS
- Khi ReduceTaskTracker thực hiện thành công reduce task, thì nó sẽ gửi thông báo trạng thái "completed" của reduce task được phân công đến JobTracker
- Nếu reduce task này là task cuối cùng của job thì JobTracker
  - Trả về cho chương trình người dùng biết job này đã hoàn thành
  - Làm sạch cấu trúc dữ liệu của mình mà dùng cho job này
  - Thông báo cho các TaskTracker xóa tất cả các dữ liệu output của các map task

Làm việc ở ReduceTaskTracker buoi3 dung



Ứng dụng của MapReduce

- MapReduce không phải là mô hình áp dụng được cho mọi vấn đề
- MapReduce áp dụng tốt cho các trường hợp cần xử lý một khối dữ liệu lớn bằng cách chia ra thành các mảnh nhỏ hơn và xử lý song song
- Một số trường hợp thích hợp với MapReduce:
  - Dữ liệu cần xử lý lớn, kích thước tập tin lớn
  - Các ứng dụng thực hiện xử lý, phân tích dữ liệu, thời gian xử lý đáng kể, có thể tính bằng phút, giờ, ngày, tháng..
  - Cần tối ưu hoá về băng thông trên cluster
- Một số trường hợp có thể không phù hợp:
  - Dữ liệu cần xử lý là tập hợp nhiều tập tin nhỏ
  - Cần tìm kiếm nhanh (tốc độ có ý nghĩa đến từng giây) trên tập dữ liệu lớn. Do độ trễ khi xử lý các MapReduce Job và khởi tạo các task trên DataNode