⁎*⁎ Processamento de informação distribuida ⁎*⁎

Motivação

- Processamento de dados em grande escala
  - com o objetivo de usar 1000s de CPUs
  - mas com gerenciamento simplificado
- MapReduce fornece
  - Paralelização automática e distribuição
  - Tolerância ao erro
  - Planeamento de I/O
  - Atualizações de monitoramento e status

MapReduce - introdução

- Solução padrão
- Fácil de distribuir através de nós
- Boa semântica de repetição / falha
- Implementações
  - Hadoop MapReduce
  - MongoDB e CouchDB

MapReduce - framework

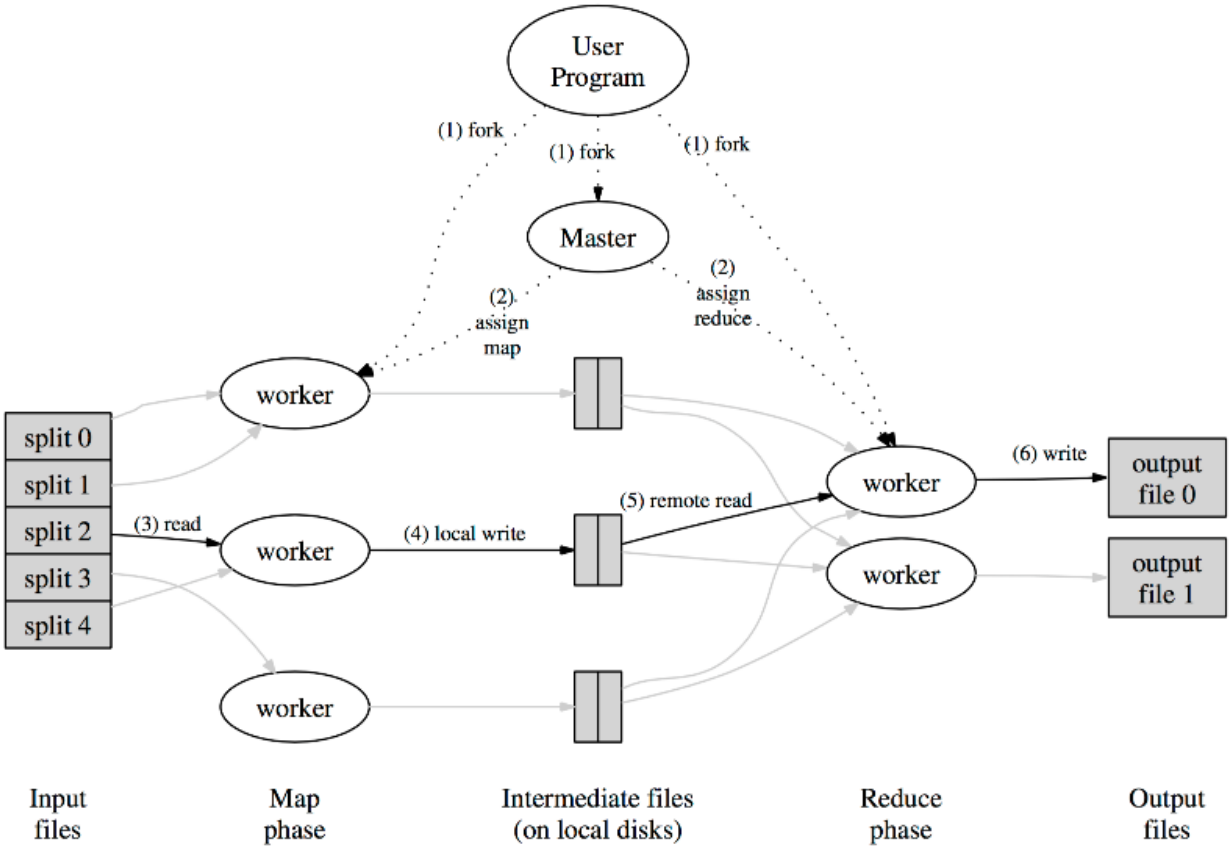
- 2 tipos de tarefas:
  - Mappers: processam os dados um determinado nó do cluster
  - Reducers: combine esses resultados de forma a resolver o problema
- A framework planeja tarefas, monitoriza-as e re-executa tarefas erradas
- Principais tarefas (pipeline)
  1. **Dividir o conjunto de dados de entrada em pedaços independentes**
  2. Mappers fazem o processamento de forma completa
  3. Sort das saídas dos mapas
  4. Reducers processam a saída de sorted maps

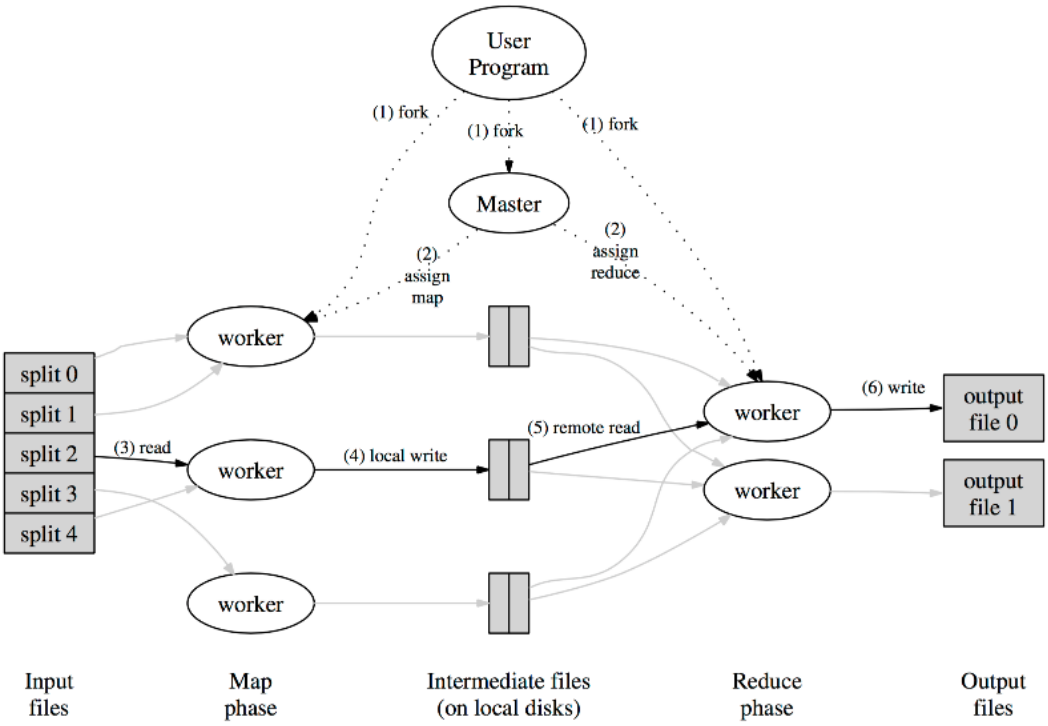
Modelo de programação

- Input & Output
  - um conjunto de pares chave / valor
- Duas funções
  - **map** (in_key, in_value) -> list(out_key, intermediate_value)
    - processa os pares chave/valor (input)
    - produz conjunto de pares intermédios
      - geralmente de um domínio diferente
      - podem existir chaves repetidas
  - **reduce** (out_key, list(intermediate_value)) -> list(out_value)
    - combina todos os valores intermediários para uma chave específica
    - produz um conjunto de valores de saída merged (geralmente apenas um)
      - do mesmo domínio

Dataflow

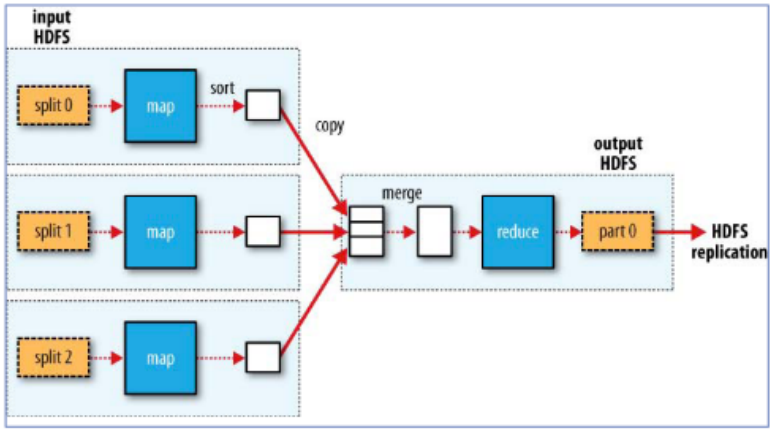
1. Arquivos / documentos de Input são divididos em records
2. A função de **Map** é chamada uma vez para cada input record, esta vai extrair (um ou mais) valor-chave do input record
3. A framework MapReduce colhe todos os pares de valor-chave com a mesma chave e ordena pela key
4. A função de reduce percorre todos os valores com a mesma key e pode produzir output records (como o número de ocorrências da chave)



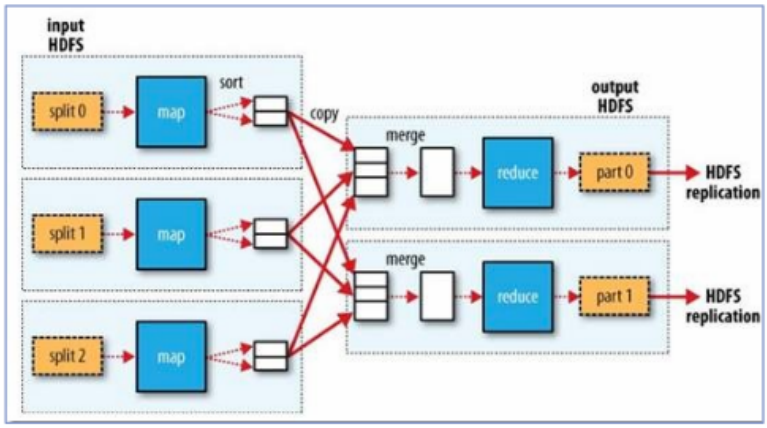


Task execution - distinct configurations

single reduce task



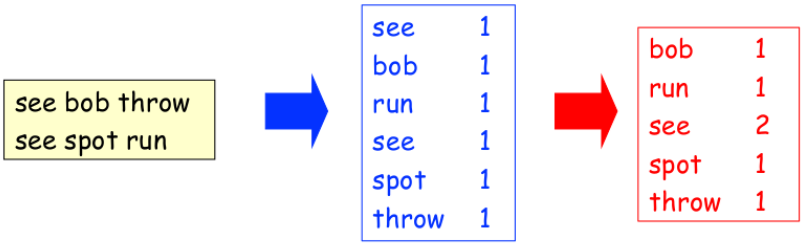
multiple reduce task



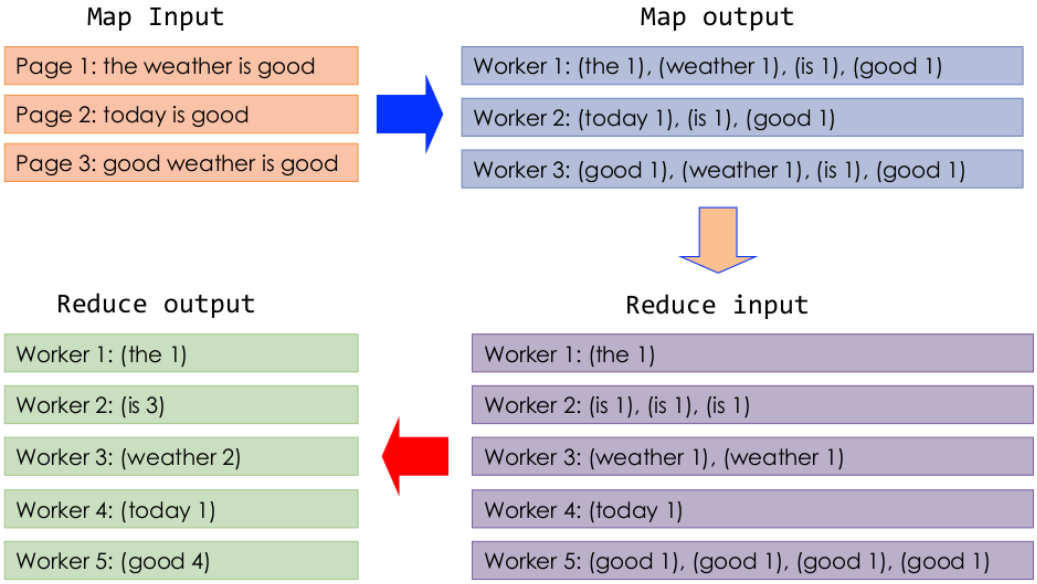
Words count

- Input
  - (url, contents) pairs

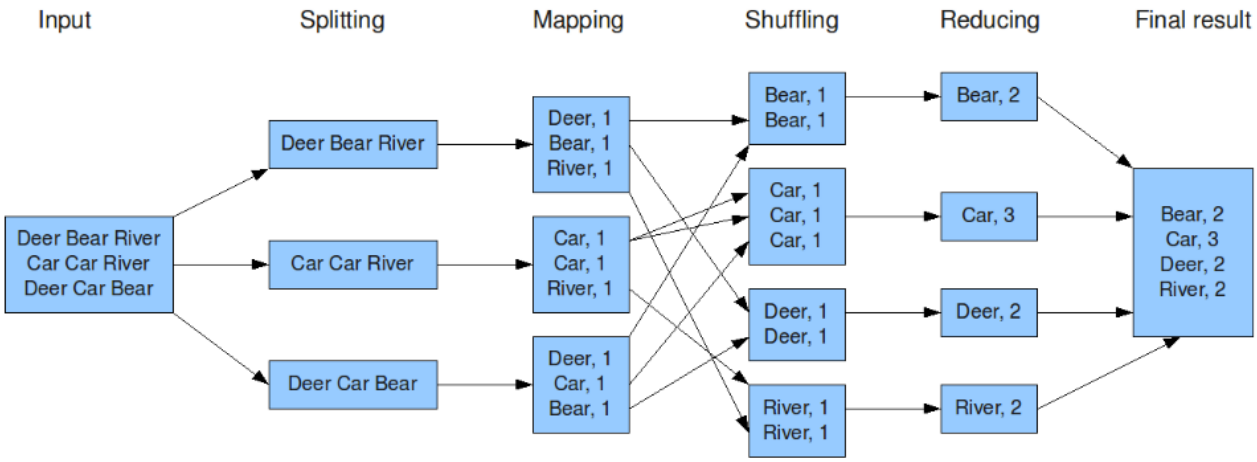
- map(key=url, val=contents):
  - for each word w in contents, emit (w, “1”)
- reduce(key=word, values=uniq_counts):
  - sum all “1”s in values list
  - emit result “(word, sum)”



Words count – tasks execution



Words count - dataflow



Words count - MapReduce function

```
// Map function
// @param key Document name
// @param value Document contents
map(String key, String value) {
    foreach word w in value: emit(w, 1);
}

//Reduce function
// @param key Particular word
// @param values List of count values associated with the word
```

```
reduce(String key, Iterator values) {
   int result = 0;
   foreach v in values: result += v;
   emit(key, result);
}
```

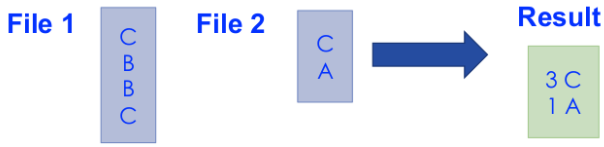
MapReduce - Aplicações

O modelo MapReduce é amplamente aplicável

- contagem de palavras / histograma
- grep distribuído
- ordenação distribuida
- web link-graph reversal
- termo-vetor por host
- estatísticas de log de acesso da web
- construção do índice invertido
- clustering de documentos
- aprendizagem de máquinas
- tradução estatística da máquina

Grep Distribuído

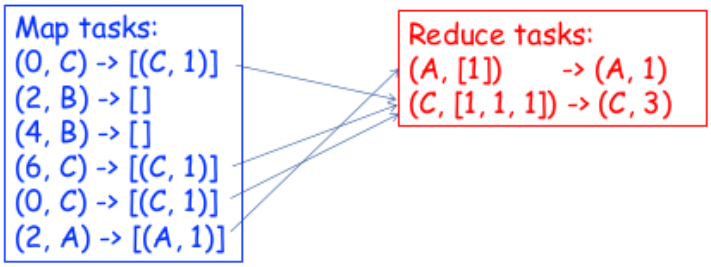
- No Unix:
  - grep -Eh <regex> <inDir> / * | classificar | uniq -c | classificar -nr
    - contagens em todos os arquivos em <inDir> que correspondem a <regex> e exibe as contagens em ordem decrescente
- Exemplo de Utilização
  - Analisando os logs de acesso do servidor web para encontrar as principais páginas solicitadas que correspondem a um determinado padrão
    - grep -Eh 'A|C' in/* | sort | uniq -c | sort -nr



❖ **Map** function
- input
  (file offset, line)
- output is either
  1. empty list [] (if line does not match)
  2. key-value pair [(line, 1)] (if line matches)

❖ **Reduce** function
- input
  (line, [1, 1, ...])
- output
  (line, n) (n is the number of 1s in the list)

Distributed grep – with MapReduce



Geração de PDF em grande escala

- O problema
  - O New York Times precisava gerar arquivos PDF para 11.000.000 de artigos (todos os artigos de 1851-1980) na forma de imagens digitalizadas a partir do papel original
  - Cada artigo é composto de inúmeras imagens TIFF que são escaladas e coladas juntas
  - O código para gerar um PDF é relativamente direto
- Tecnologias utilizadas
  - Amazon Simple Storage Service (S3)
    - Armazenamento de internet escalável e barato que pode armazenar e recuperar qualquer quantidade de dados a qualquer momento de qualquer lugar da web
    - Sistema assíncrono e descentralizado que visa reduzir os bottlenecks de escala e os únicos pontos de falha
  - Amazon Elastic Compute Cloud (EC2)
    - Ambiente de computação virtualizado projetado para uso com outros serviços da Amazon (especialmente S3)
  - Hadoop
    - Implementação de código aberto do MapReduce
- Resultados
  - 4TB de artigos escaneados foram enviados para S3
  - Um conjunto de máquinas EC2 foi configurado para distribuir a geração de PDF via Hadoop
- Utilizando 100 instâncias de EC2 e 24 horas, o New York Times foi capaz de converter 4TB de artigos digitalizados para 1.5TB de documentos PDF