百度系统部分布式系统介绍

马如悦
maruyue@baidu.com
2010.08.27

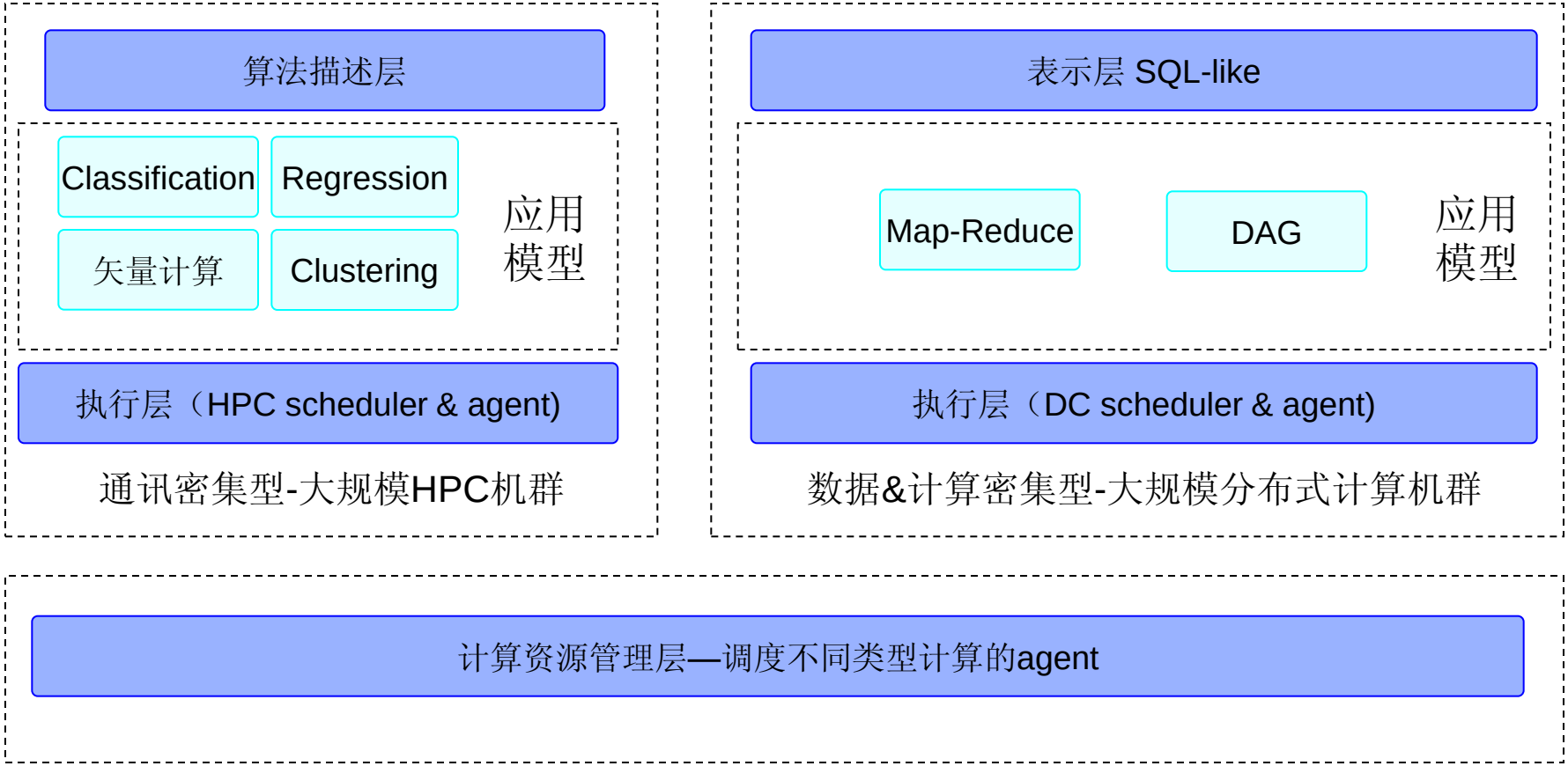
主要内容

- 百度的数字
- 计算平台
  - HPC
  - DC
- 存储平台
  - DFS
  - DOS
- 数据平台(?)

百度的数字

- HPC - 高性能计算
  - 200左右，8 core, 16GB~64GB
- DC - MapReduce计算
  - > 10个集群，共4000台
  - 8core，16GB，12*1TB
  - 最大集群1000台
  - 每日计算量>2.5PB
  - 每日作业数>3w
- DS - 分布式存储
  - 使用容量平均70%

计算平台



HPC

- 单机HPC
  - Multicore
  - GPU
  - FPGA
- 多机HPC
  - MPI
- 机器学习算法服务
  - 单机HPC+多机HPC+MapReduce
- 应用
  - 商务搜索、Baidu News ……

DC

- Abaci = Hadoop-MapRed+自主系统
- 表示层
  - DISQL (to appear in Hadoop in China 2010)
- 调度层
  - Master+Agent
  - workflow元调度(in Master)
  - 数据分发服务(in Agent): shuffle，bt文件分发
  - 分布式Master
- 计算层
  - Streaming - 文本处理
  - Bistreaming - 二进制处理
  - HCE - C++编程接口
  - ……

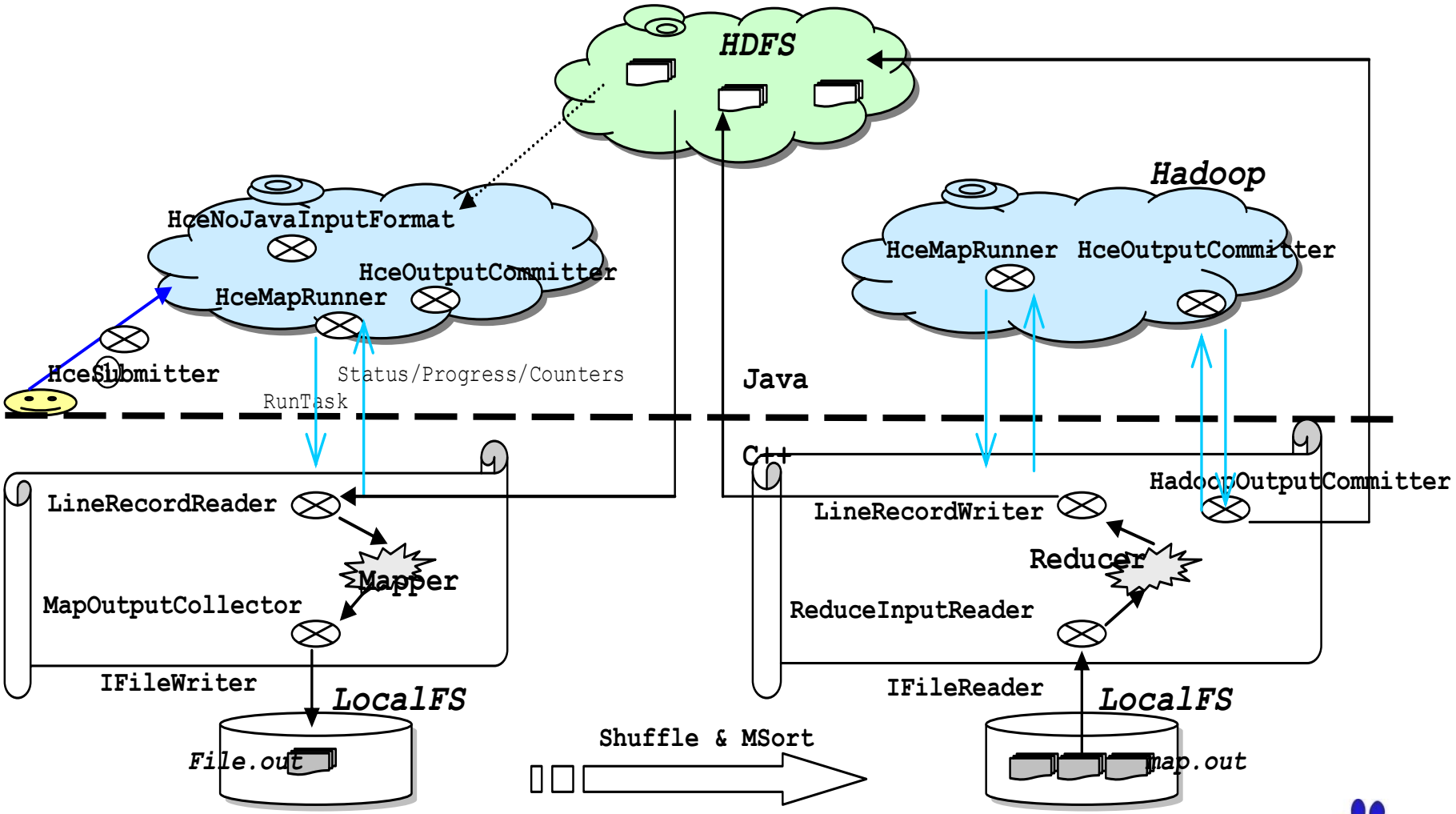
DC-HCE-介绍

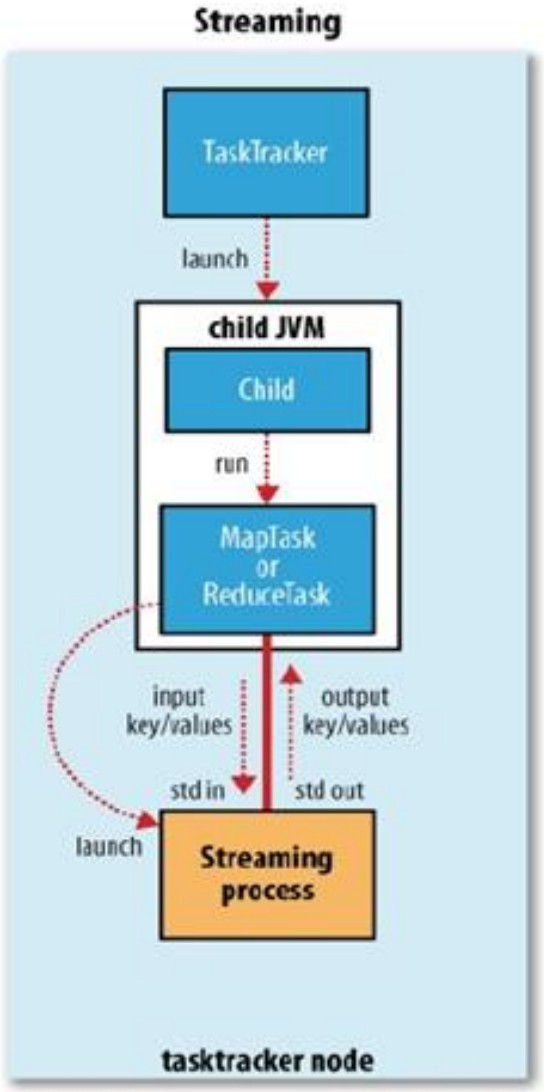
- Hadoop C++ Extension
- Jira: MapReduce-1270
  - https://issues.apache.org/jira/browse/MAPREDUCE-1270
  - Design Doc
  - Patch
  - Demo package
  - Install Manual
  - Tutorial
  - Performance Test Doc

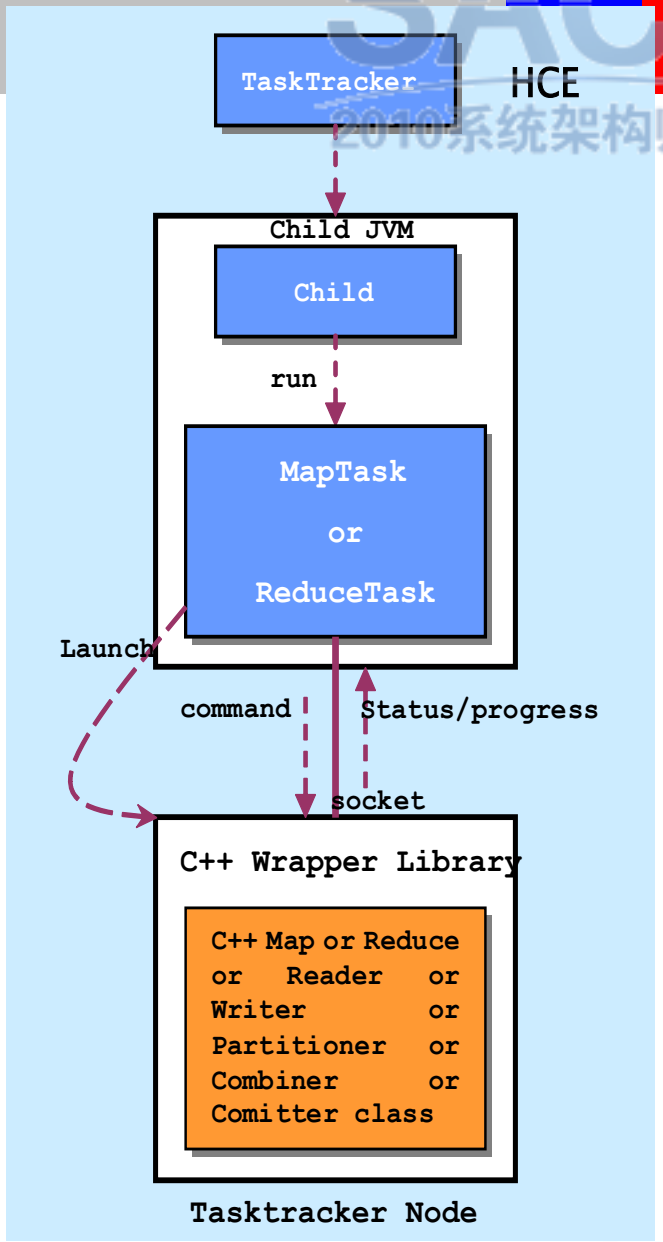
DC-HCE-背景

- Why not Pipes, Bistreaming
- Java语言效率: 提升10%~40%
  - sort, compress/decompress
- Java内存控制
- Full featured C++ API

DC-HCE-Design







DC-HCE-基本接口


```
class Mapper {
public:
  virtual int64_t setup() = 0;
  virtual int64_t cleanup() = 0;
  virtual int64_t map(MapInput &input) = 0;

protected:
  void emit(const void* key, int64_t keyLength,
            const void* value, int64_t valueLen
  TaskContext* getContext();
};
```

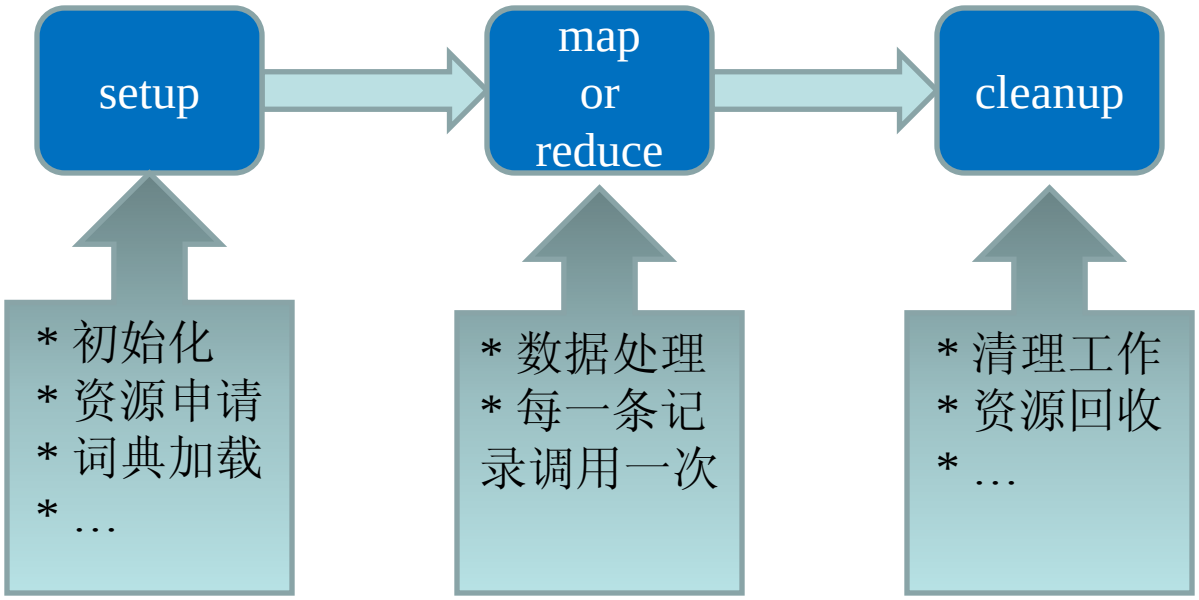
```
class Reducer {
public:
  virtual int64_t setup() = 0;
  virtual int64_t cleanup() = 0;
  virtual int64_t reduce(ReduceInput &input) = 0;

protected:
  void emit(const void* key, int64_t keyLength,
            const void* value, int64_t valueLength);
  TaskContext* getContext();
};
```

setup(), cleanup(), map(), reduce()函数必须实现，函数的返回值是执行结果，0表示正常，非0表

提供给用户调用的函数，emit()函数用来输出数据，Context()函数得到TaskContext

DC-HCE-接口调用顺序



DC-HCE-Demo


```
class WordCountMap: public HCE::Mapper {
  public:
    int64_t setup() {
      return 0;
    }
    int64_t map(HCE::MapInput &input) {
      int64_t size = 0;
      const void* value = input.value(size);
      if ((size > 0) && (NULL != value)) {
        char* text = (char*)value;
        const int n = (int)size;

        for (int i = 0; i < n;) {
          // Skip past leading whitespace
          while ((i < n) && isspace(text[i])) i++;
          // Find word end
          int start = i;
          while ((i < n) && !isspace(text[i])) i++;
          int count = 1;
          if (start < i) {
            emit(text + start, i-start, &count, sizeof(int));
          }
        }
      }

      return 0;
    }
    int64_t cleanup() {
      return 0;
    }
};
```

```
class MapInput {
public:
    const void* key(int64_t& size) const;
    const void* value(int64_t& size) const;
};
```

DC-HCE-Demo


```
class WordCountReduce: public HCE::Reducer {
  public:
    int64_t setup() {
      return 0;
    }

    int64_t reduce(HCE::ReduceInput &input) {
      int64_t keyLength;
      const void* key = input.key(keyLength);

      int64_t sum = 0;
      while (input.nextValue()) {
        int64_t valueLength;
        const void* value = input.value(valueLength);
        sum += *((const int*)value);
      }

      const int INT64_MAXLEN = 25;
      char str[INT64_MAXLEN];
      int str_len = snprintf(str, INT64_MAXLEN, "%ld", sum);
      emit(key, keyLength, str, str_len);
    }

    int64_t cleanup() {
      return 0;
    }
};
```

```
class ReduceInput  {
public:
    const void* key(int64_t& size);
    const void* value(int64_t& size);
    bool nextValue();
};
```

DC-HCE-Other

接口	功能
Combiner	在Map完成后进行局部规约 默认没有Combiner操作
Partitioner	中间数据分桶函数，根据Key确定由哪一个Reduce来处理 默认是HashPartitioner，按照Key的Hash值进行分桶
RecordReader	数据输入接口。默认是LineRecordReader，用于读取文本数据。HCE另外提供SequenceRecordReader可以读取SequenceFile格式的二进制数据
RecordWriter	数据输出接口。默认是LineRecordWriter，用于输出文本数据。HCE另外提供SequenceRecordWriter可以输出SequenceFile格式的二进制数据
JobConf	读取Hadoop环境配置和用户自定义配置
Counter	进行作业级别的数据统计

DS

- Distributed Store
- HDFS2 - DFS
  - Scalability - DistributedNameNode(DNN)
  - Availability
  - Low latency
  - High concurrent
- ? - DOS
  - 无树状命名空间，二层命名空间
  - Object: 几KB->几GB
  - REST API

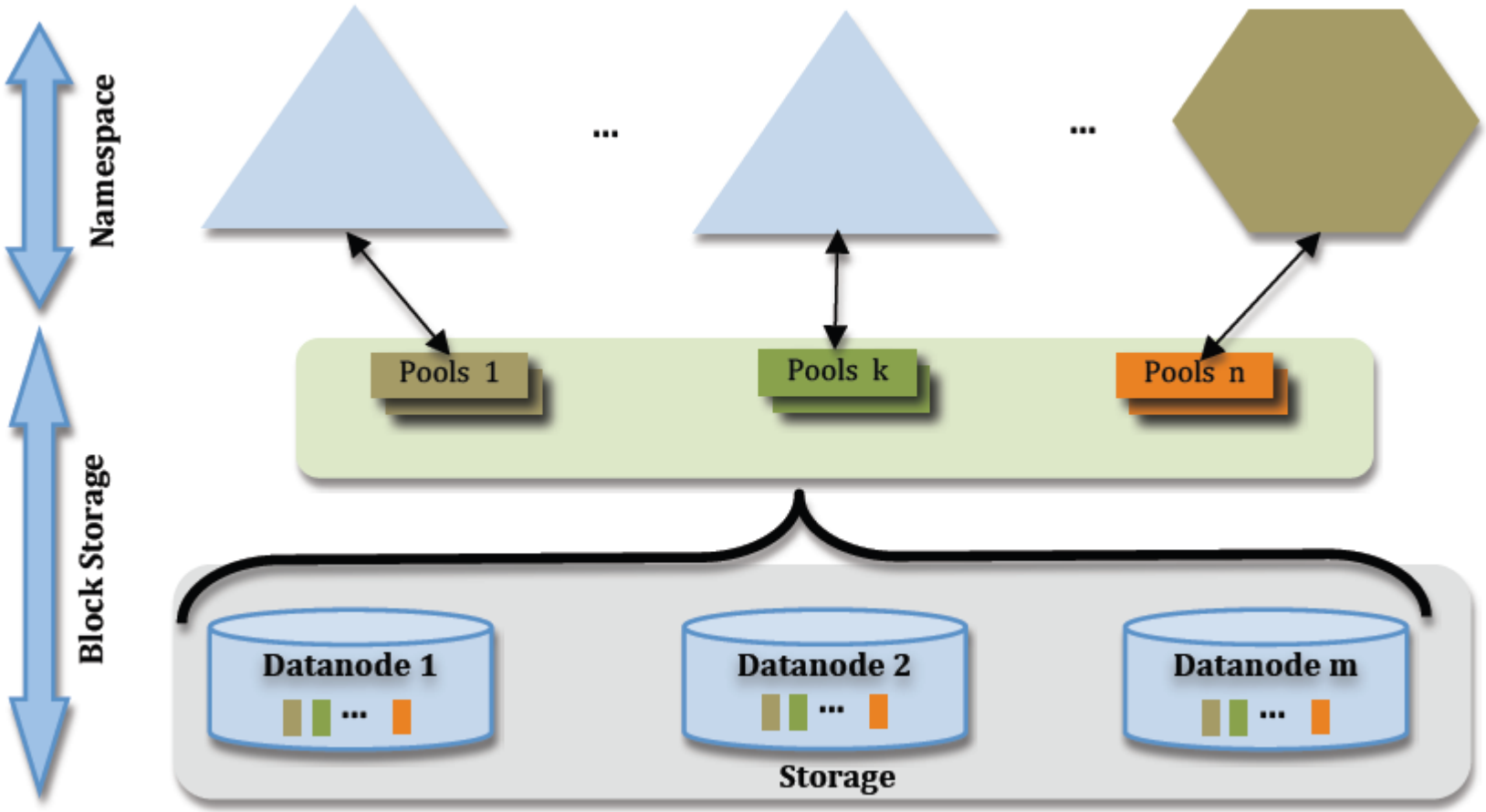
HDFS2-DNN-背景

- 31,855,959个文件，34,233,901个块
  - 内存占用约12GB
    - 块管理 ≈ 7.8G，包括全部块副本信息
    - 目录树 ≈ 4.3G，目录层次结构，包含文件块列表信息
- 到10亿文件、10亿块的数据规模
  - 内存占用约380GB
    - 块管理 ≈ 240GB
    - 目录树 ≈ 140GB
- 负载
  - 集群规模扩大后，单点的NameNode请求压力也会同时增大

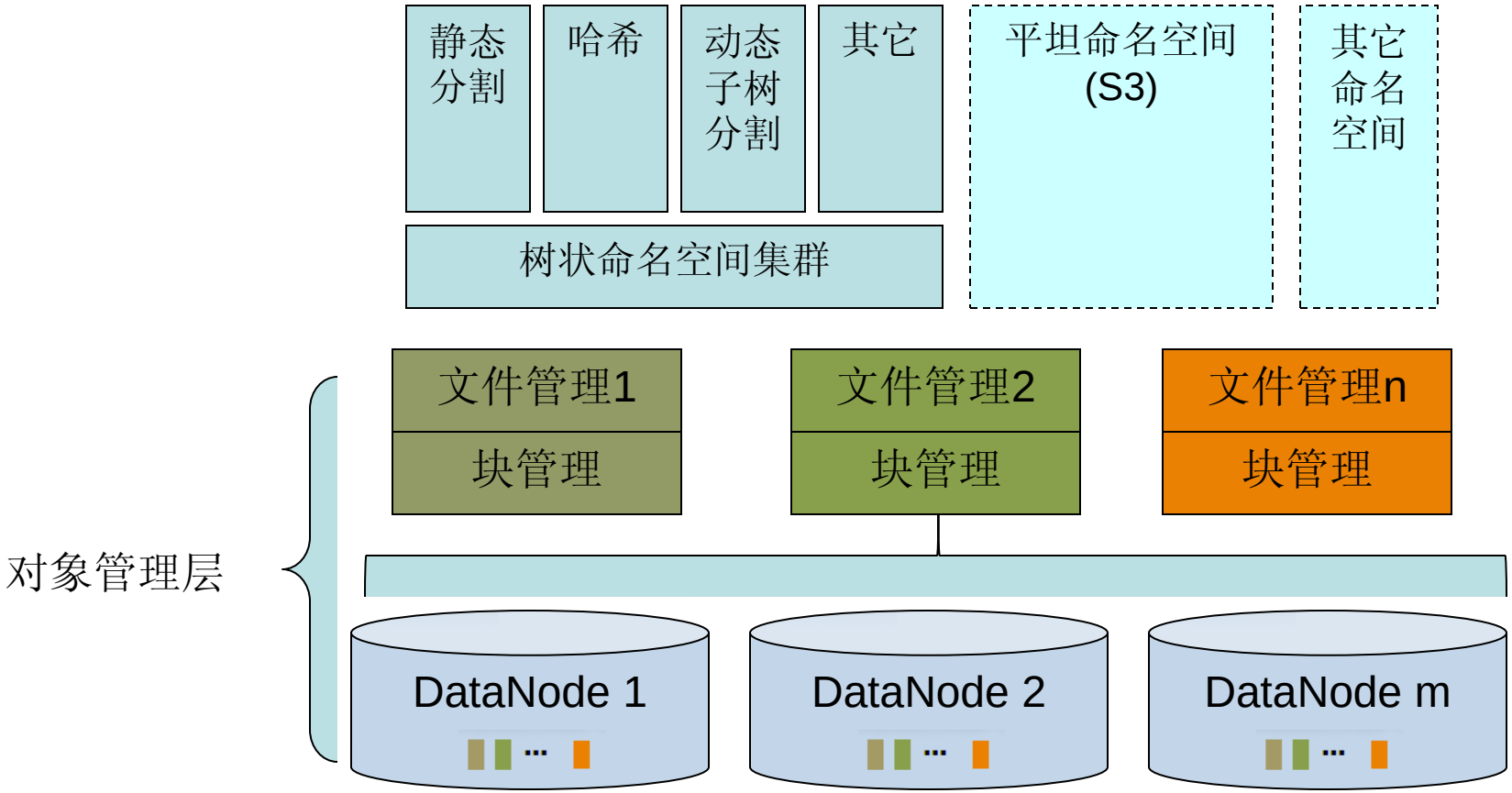
HDFS2-DNN-调研

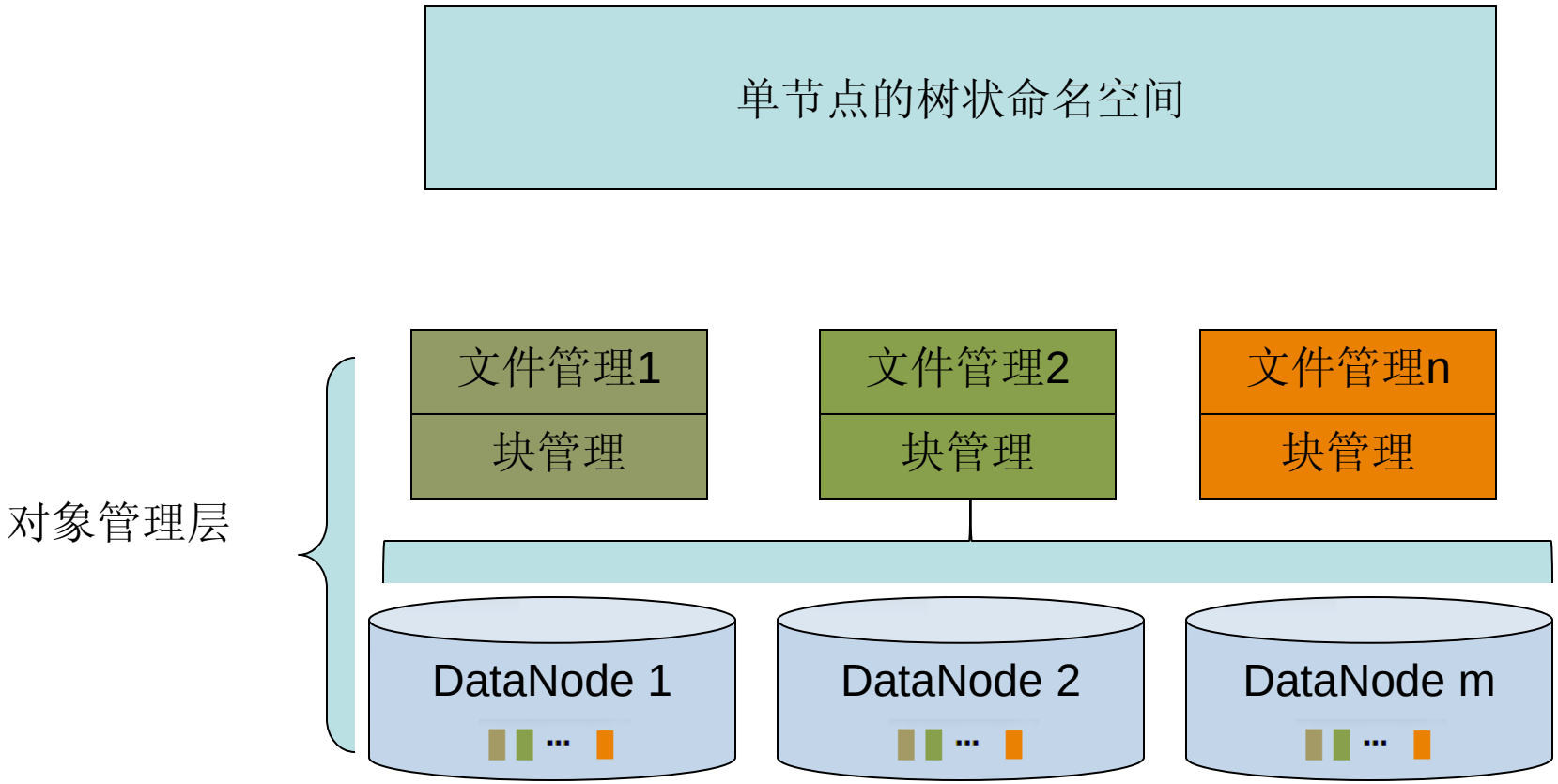
	Lustre	Ceph	GFS2
分布式元数据组织策略	•随机、轮询等 •各节点平坦存放INode信息	•动态子树分割，元数据服务维护INode到根目录的路径 •子树更新机制，修改可以同步到每一个节点上的子树副本	
独立的块、对象管理层	是	是，独立的对象管理层RADOS，通过P2P维护内部的数据一致性和副本安全	推测是
元数据存储策略	共享对象存储	共享对象存储	共享存储Bigtable
元数据定位	•路径查询和权限控制需要根据目录层级遍历多个元数据服务。 •解决性能问题采用了客户端缓存和分布式锁机制	无需跨多台	
特色		RADOS、CRUSH	•支持小文件存储 •支持低延迟的交互式请求

HDFS2-DNN-社区方案



HDFS2-DNN-Ours




单节点的树状命名空间
对象管理层
文件管理1
块管理
文件管理2
块管理
文件管理n
块管理
DataNode 1
DataNode 2
DataNode m

HDFS2-DNN-优点

- 文件对象管理服务直接就是水平可扩展的
- 文件对象管理做为单独服务存在，可挂载不同类型命名空间，如S3
- 大幅度减轻了Namespace的职责，方便后续对Namespace进行分布式化
- 间接减少了Namespace的压力
  - 很多数据不用再存储在Namespace上
  - 很多请求不用再通过Namespace

HDFS2-DNN-预期

- 内存
  - 10亿文件，10亿块
    - 文件 ≈ 66GB，目录 ≈ 1GB
    - 单节点命名空间就可以管理
- 请求负载
  - 大部分的耗时操作都属于文件对象管理层，不用经过Namespace
  - 最耗CPU资源的若干操作中，仍需经过Namespace的只占13.7%
  - 命名空间管理不再维护块信息，大部分操作都不需要加全局锁，可以更充分利用CPU资源