网易基于 Elasticsearch 构建通用搜索系统的实践

陈亮亮
2019/12/7，搜索平台负责人, 网易杭州研究院

目录

平台经验与总结

多路召回及算法集成框架

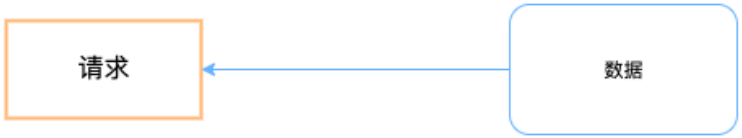
多路召回概念

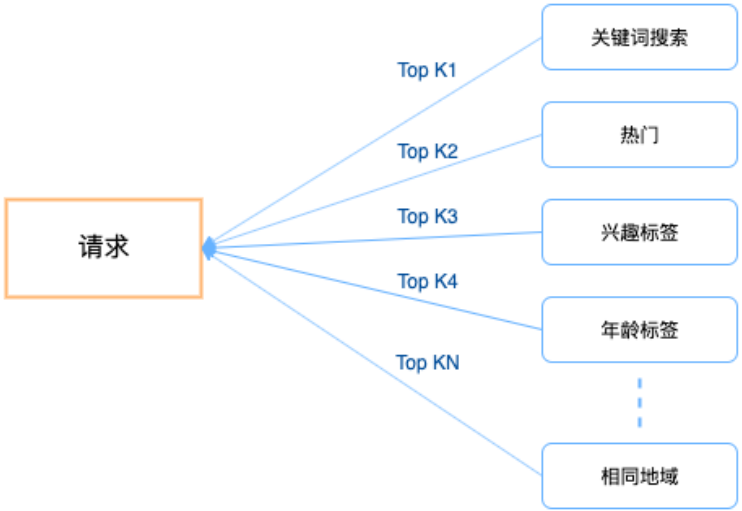
1.
一个请求扩展为多个子查询信息，每个子请求作为一路，取回多个查询结果

2.
每一路召回可能采取一个不同的策略

3.
各路查询条件一般不能相互影响

4.
如果分成多个单次召回，性能损耗太大





Elasticsearch 现有解决方案

组合查询 Compound Queries

- bool query (must, should, must_not, filter)
- dis_max query (single best-matching query)
- other...

批量查询 Multi-Search API

- GET /<index>/_msearch
- header\n
  body\n
  header\n
  body\n

Elasticsearch 现有解决方案存在的问题

- 各查询子条件之间存在相互影响
- 很难控制最终结果排序结果值
- 可能会漏掉一些关键结果数据

- 返回结果集过大
- 在子查询过多（几十路）、每个子查询结果要求多（>=5000）的情况下性能较差

针对现有问题的改进方案

改进思路 1

一、改进功能：

融合 Compound Queries 和 Multi-Search 优点，实现独立各路查询，避免互相影响，结果又可以按需过滤合并

改进思路 2

二、算法下沉：

把算法逻辑下放到近数据端。不仅可以提供算法集成功能方便用户实现效果优化，也可以提前处理掉无效数据

最终产出

多路搜索（k-way search）：

多路召回功能 + 算法集成框架

如何在 Elasticsearch 中实现

多路召回

一、新增 Elasticsearch API —— ksearch

- 实现要点：
  1. Elasticsearch Client
     - 支持组合多个查询请求，每个查询为 es 原生查询类型
     - 支持配置多路搜索参数
     - 客户端 SDK 增加支持 ksearch api，用户可以和其他 es api 一样方便使用
  2. Elasticsearch Server
     - 读取和解析客户端传入的 ksearch 请求及相关参数
     - 实现 API 相应的 Action（Request/Response）
     - 并发执行多路请求，最大化提高性能

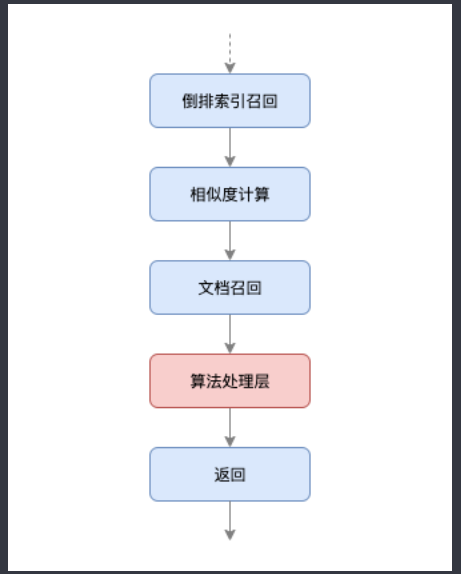
如何在 Elasticsearch 中实现

算法下沉

二、增加算法集成框架

- 实现要点：
    1. Elasticsearch Server
        - 拦截多路请求返回结果
        - 增加算法处理层（集成用户自定义算法）
        - 通过动态 load jar 的方式动态的加载用户算法 lib

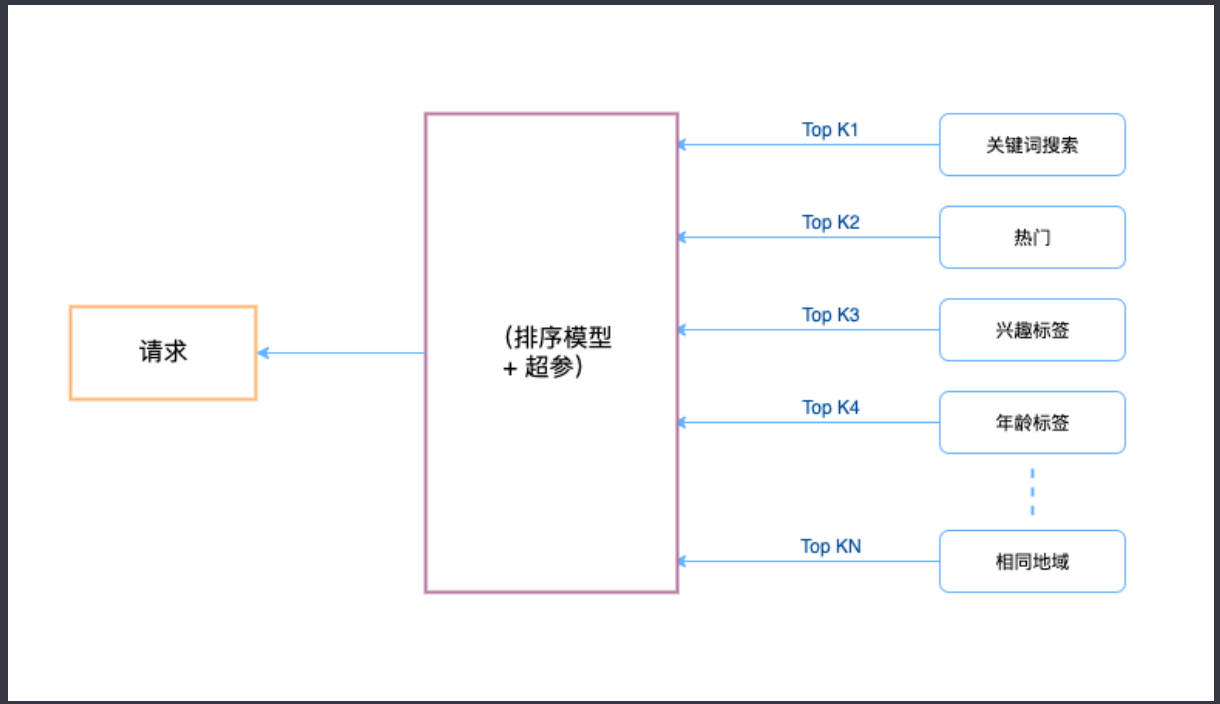
          —— es 节点无需重启，支持热更新
        - 配合多路搜索参数（超参），对每个请求可以应用不同效果



elastic

多路召回及算法集成框架

最终形式



elastic

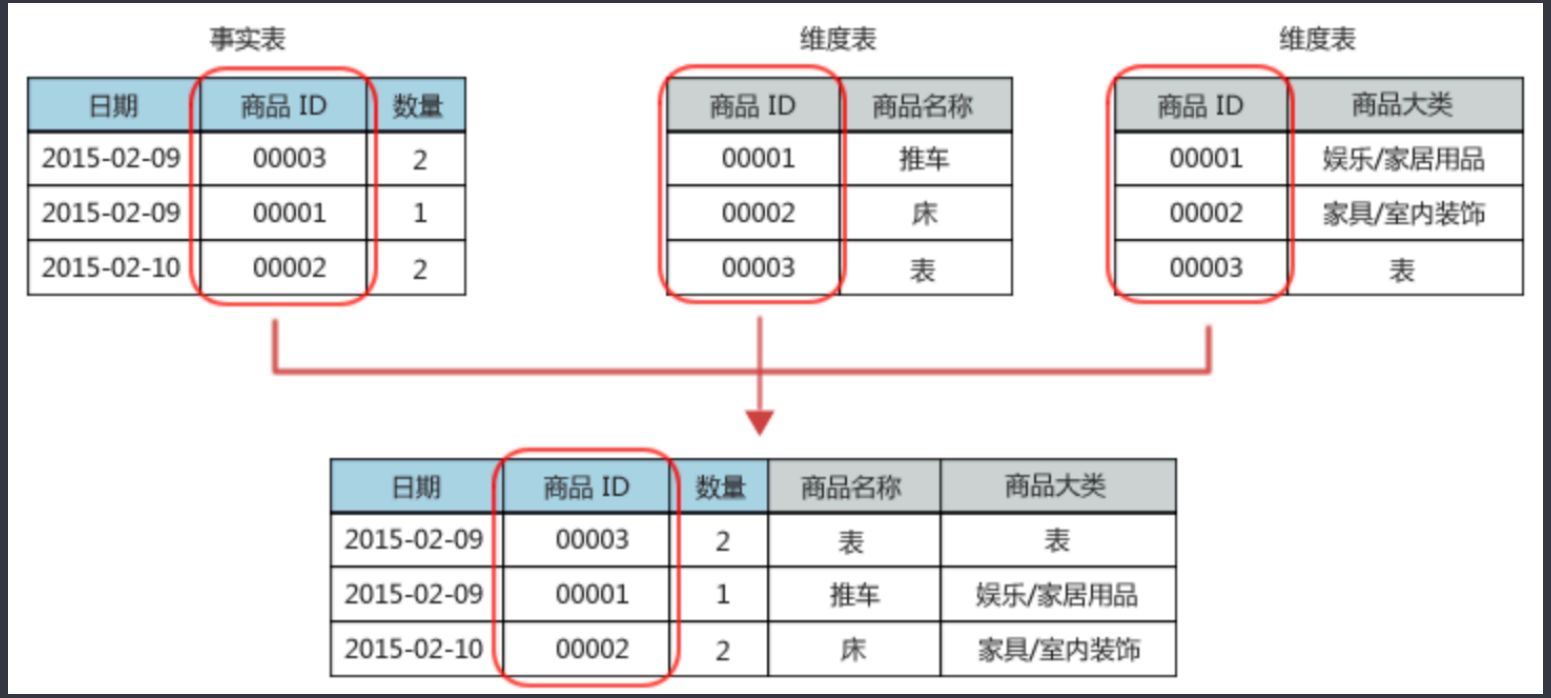
多路召回及算法集成框架

改进所获得的效果

- 极大的提高性能
- 用户可以对每一路召回更多结果，大幅提高结果质量
- 系统得以支撑更多的业务场景

数据同步中的多数据源 join 问题

什么是数据 join ?

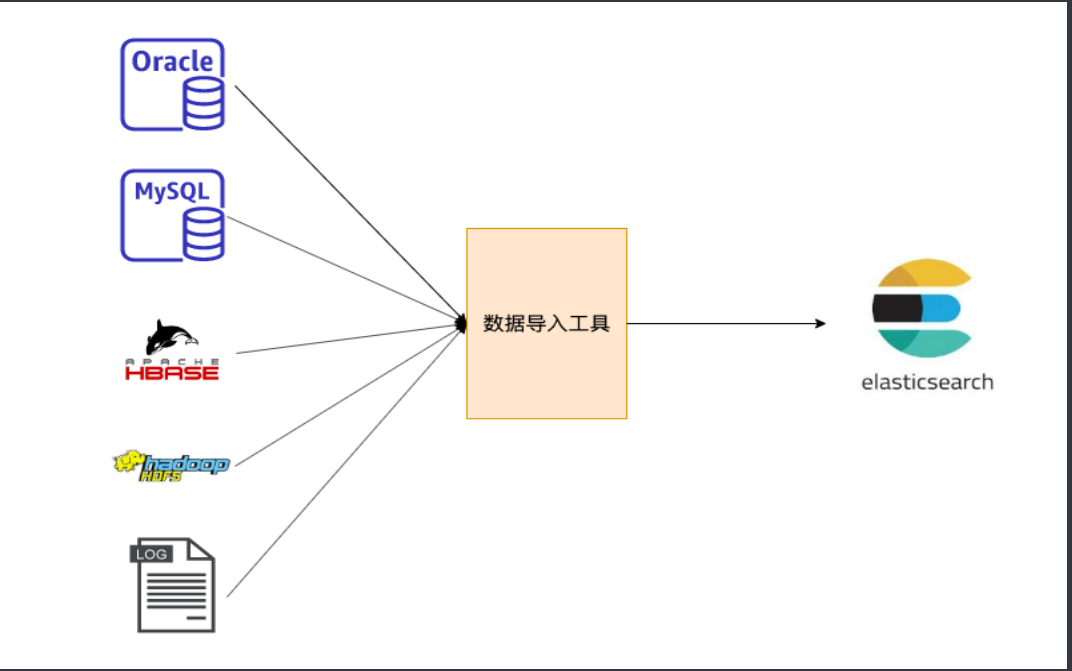


©图片来源网络

elastic

数据同步中的多数据源 join 问题

多数据源的 join



elastic

数据同步中的多数据源 join 问题

为什么需要做多数据源的 JOIN？

1. 用户数据分散（多个数据源、多个表...）
2. 数据之间往往存在着 join 关系，或者某些数据信息需要补足
3. Elasticsearch 不能很好的支持 join 操作
   - Parent/Child join 功能/性能有所欠缺

如何解决？

- 利用 Elasticsearch 高性能特性，适当允许数据冗余
- 将数据关系打平，多端数据共组大宽表

数据同步中的多数据源 join 问题

我们的实现方案

1. 利用流计算系统为基础，统一构建数据导入平台
2. 增加流式维表 join 功能，自动化处理用户数据 join 关系

数据库导入 es 流程架构

数据同步中的多数据源 join 问题

各组件介绍

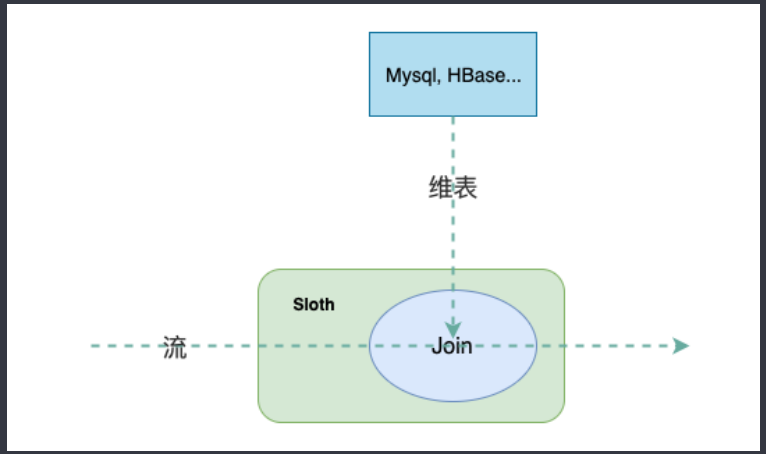
- NDC：
  1. 实时跟踪数据库 binlog 变更
  2. 详细记录与同步数据变更类型 —— 新增、修改、删除，以便目的端采取同样的操作，这是保证数据一致性的前提
- Kafka：
  1. 缓冲层 —— 削峰填谷，避免过大流量直接冲击 es
  2. 数据归集平台 —— 所有 db 数据归集至统一地方，简化上下游的关系链路
  3. 缓存的数据有助于做数据恢复，Kafka 支持的重复消费是一个非常好用的功能

数据同步中的多数据源 join 问题

各组件介绍

- Sloth：

1. 基于 Flink 的流式计算平台
2. 承载导入数据的 ETL 逻辑
3. 实现多数据源/多数据表的 join 功能



- ➢ 自主研发维表 join 功能

  （支持 SQL 语法，用户只需编写 SQL 任务即可使用）

- ➢ 其中一个源表的变更操作当做实时事件流，再去 join 另一个表/数据源

- ➢ 目前已支持 join 数据源：Mysql、Hbase、Redis 等

数据同步中的多数据源 join 问题

具体实现

- 写入 ES（es sink）：

    1. 根据上游数据变更类型新增、修改、删除执行相应操作
    2. 把数据库表的 id 同步做为 es 索引的 id，可以有效降低一致性实现的复杂度

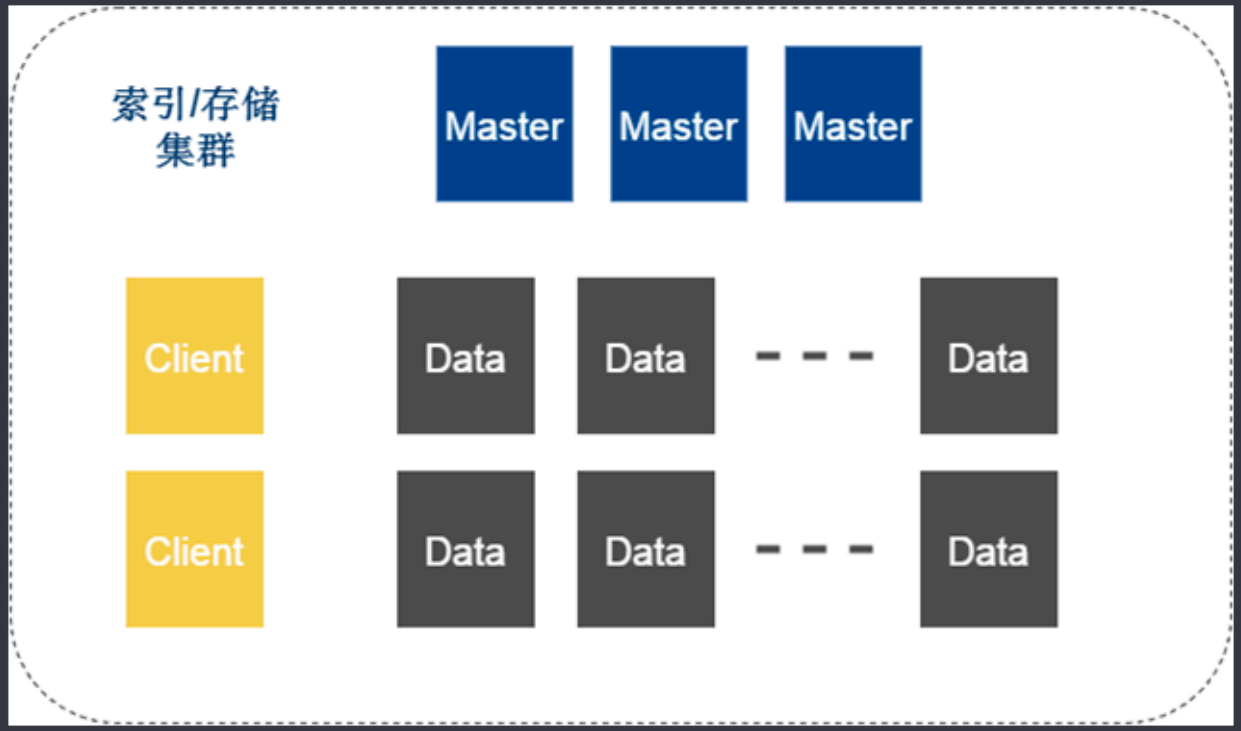
数据同步中的多数据源 join 问题

维表 join 方式实现方案的益处

- 用户无需自己额外处理 join 关系，只需简单提交一个 SQL 任务即可，由平台负责所有后续的数据处理流程
- 系统实现上，利用各组件所拥有的特性，可以便捷的实现一致性、实时性、高可用等功能
- 将数据同步统一为一个平台处理，简化用户的接入流程及系统维护成本

双机房高可用方案

线上高可用集群部署拓扑结构



elastic

双机房高可用方案

现状及问题

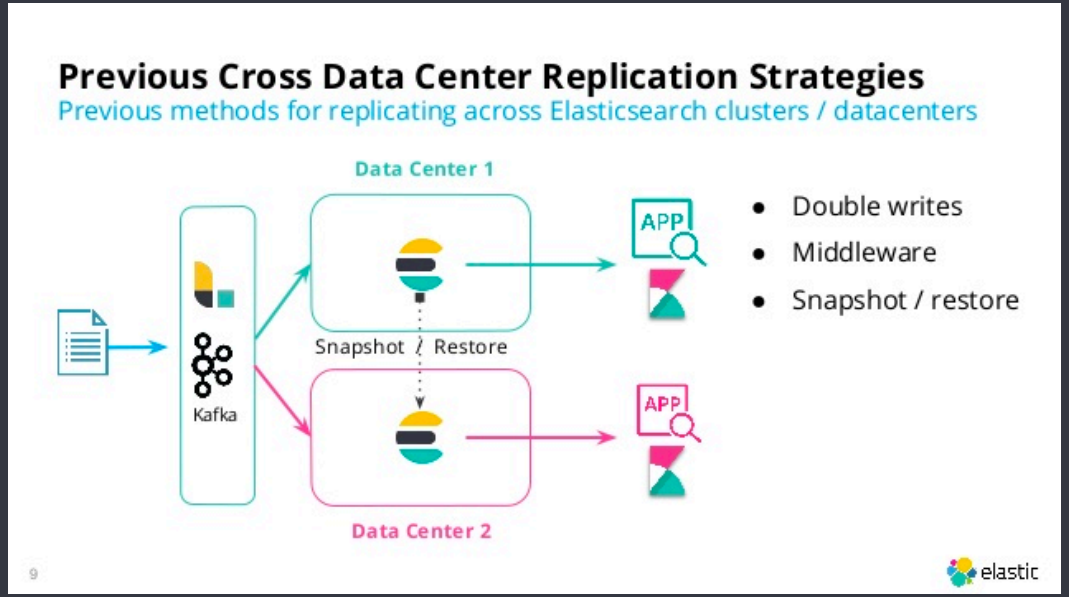
现有常见双机房高可用方案：

1. 各机房独立集群部署
2. 双写
3. 通过其他中间件进行

   跨机房数据同步
4. 使用快照恢复机制
5. es 官方跨集群复制（CCR）

存在的问题：

- 主从集群的数据一致性 / 实时性



©图片来源网络

elastic

双机房高可用方案

期望 1	期望 2	期望 3
• 单集群跨机房部署 • 多个集群往往容易导致数据不一致，实时性不一致问题	• 不引入外部服务（比如数据同步中间件） • 保持集群结构简单	• 保持数据实时性和一致性与原生 es 集群完全一致

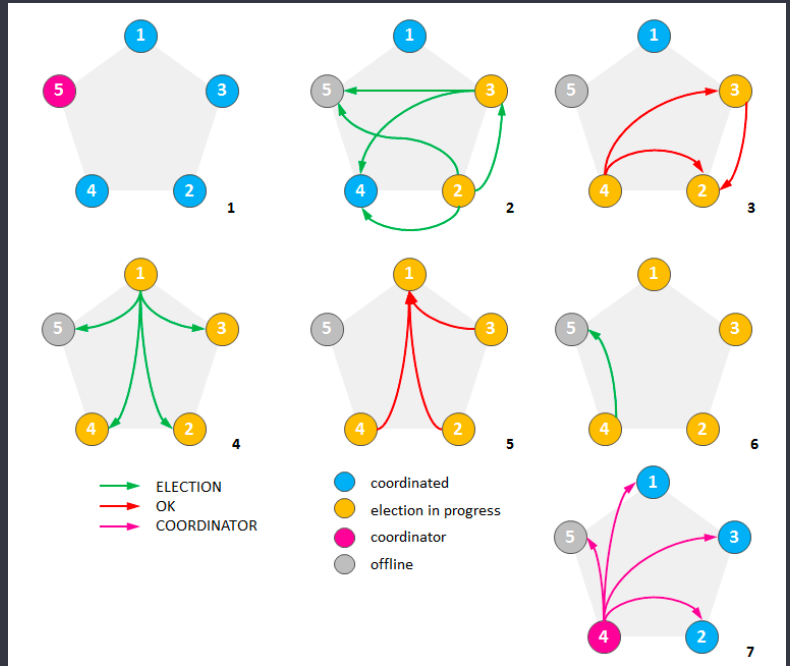
双机房高可用方案

Elasticsearch Zen-Discovery

Bully Algorithm

- Each process has a unique numerical ID
- Processes know the Ids and address of every other process
- Communication is assumed reliable
- *Key Idea*: select process with highest ID
- Process initiates election if it just recovered from failure or if coordinator failed
- 3 message types: *election, OK, I won*
- Several processes can initiate an election simultaneously
  - Need consistent result
- $O(n^2)$ messages required with n processes

UMASS Computer Science CS677: Distributed OS Lecture 12, page 4



elastic

双机房高可用方案

Elasticsearch Zen-Discovery

```java
/**
 * compares two candidates to indicate which the a better master.
 * A higher cluster state version is better
 *
 * @return -1 if c1 is a batter candidate, 1 if c2.
 */
public static int compare(MasterCandidate c1, MasterCandidate c2) {
    // we explicitly swap c1 and c2 here. the code expects "better" is lower in
    // list, so if c2 has a higher cluster state version, it needs to come first
    int ret = Long.compare(c2.clusterStateVersion, c1.clusterStateVersion);
    if (ret == 0) {
        ret = compareNodes(c1.getNode(), c2.getNode());
    }
    return ret;
}
```

1.

2.

双机房高可用方案

Elasticsearch Zen-Discovery

```
/** master nodes go before other nodes, with a secondary sort by id **/
 private static int compareNodes(DiscoveryNode o1, DiscoveryNode o2) {
    if (o1.isMasterNode() && !o2.isMasterNode()) {
        return -1;
    }
    if (!o1.isMasterNode() && o2.isMasterNode()) {
        return 1;
    }
    return o1.getId().compareTo(o2.getId());
}
```

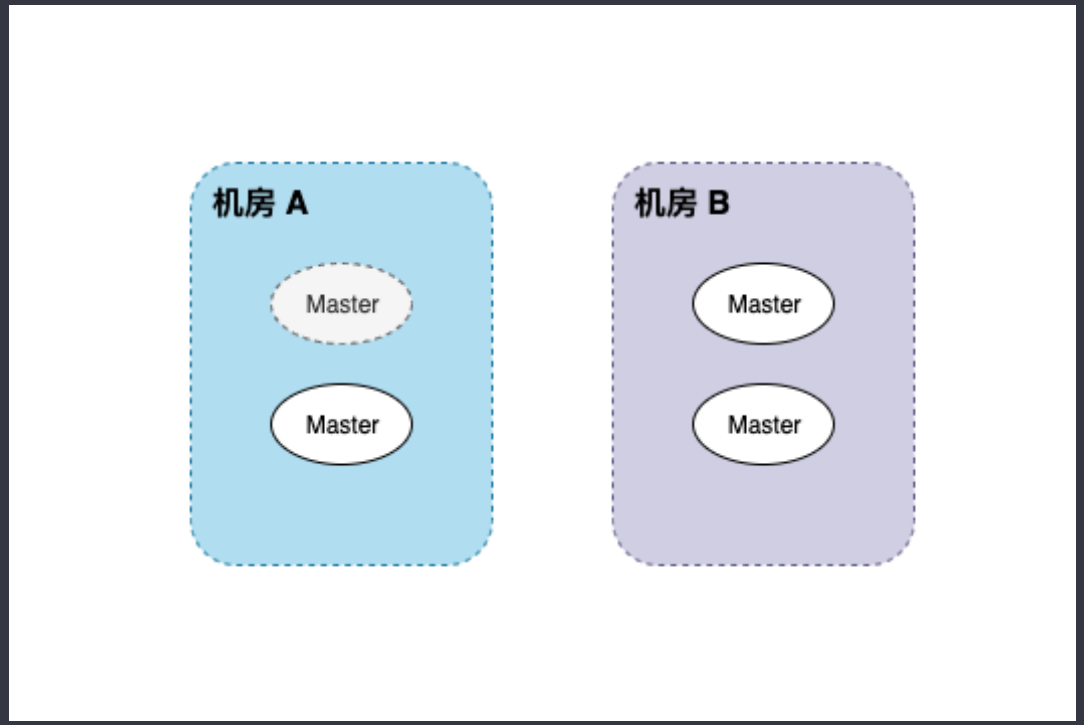
3.

4.

双机房高可用方案

选主改进

- Master 节点分 2 组
- 1：2 放置
- 机房 A 预留一个备用 master 节点（不启用）

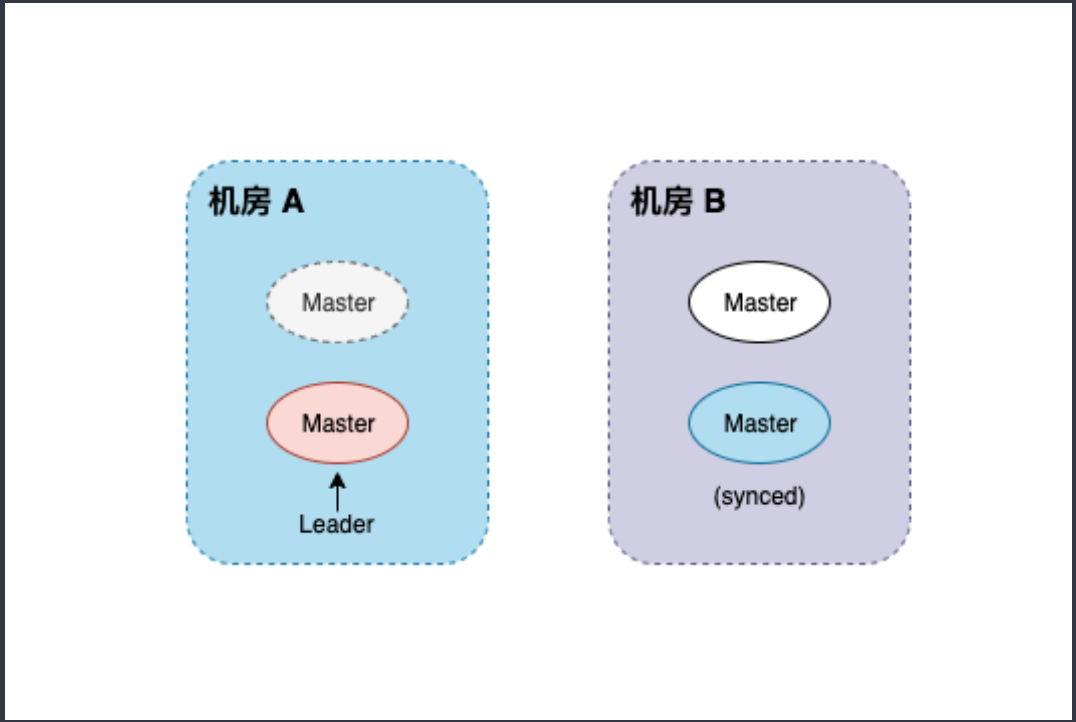


elastic

双机房高可用方案

选主改进

- 确保单个节点机房 master 被选为 Leader（控制节点 ID）
- 根据 es 的 quorum 机制，另一机房必有一同步节点



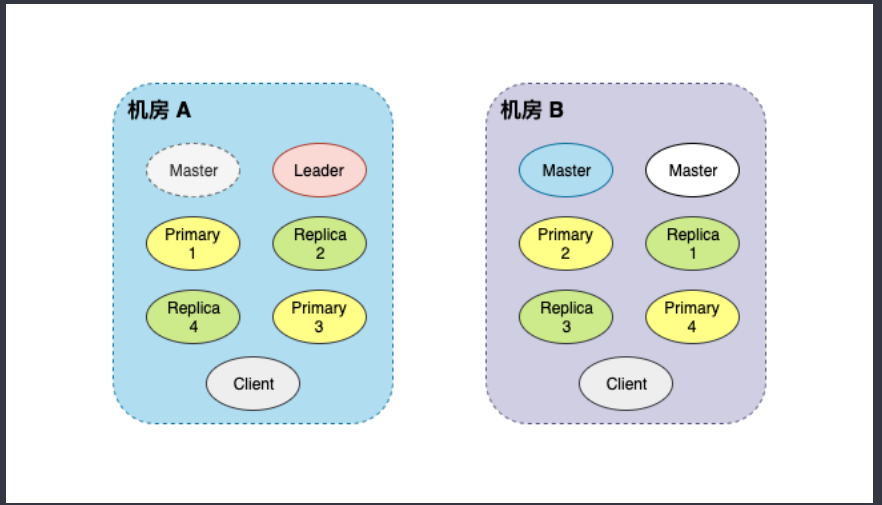
elastic

双机房高可用方案

选主改进

Data 节点分布：

- 确保节点均匀分布到 2 机房
- 确保主分片和副本分片均匀分布到 2 个机房（通过 es 选项配置）



Client 节点分布：

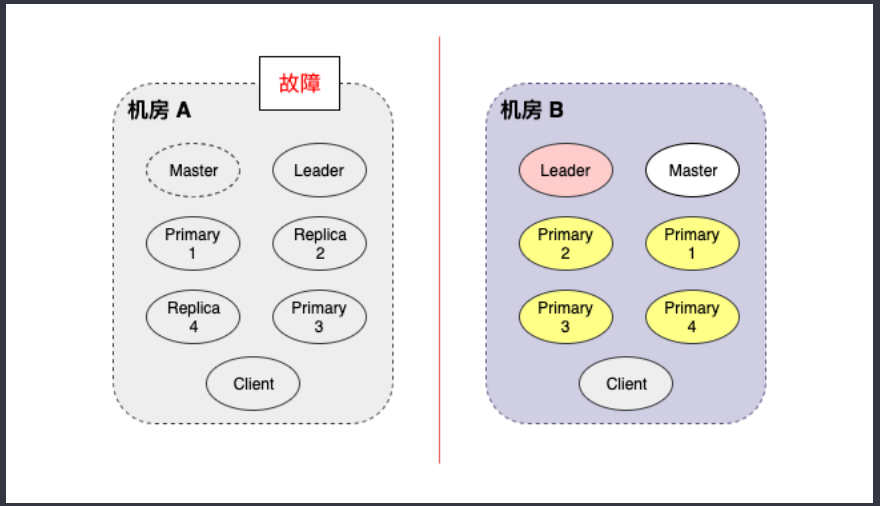
- 无状态，只需确保每个机房至少一个
- 可配置优先本地机房查询

双机房高可用方案

故障恢复

机房 A 异常：

- 机房 B 中 Master 节点过半数
- 拥有最新数据的候选节点选为 Leader
- 数据主从分片自动切换
- 客户端依然可以通过机房 B Client 节点访问



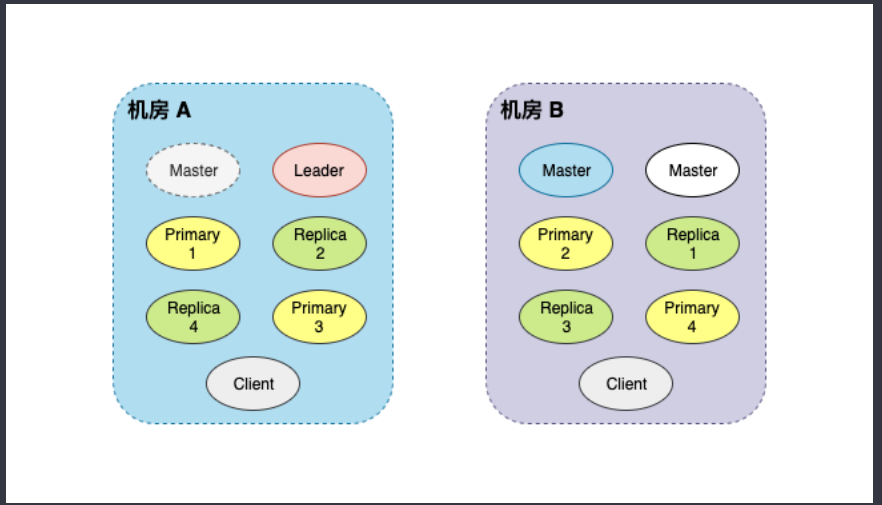
elastic

双机房高可用方案

故障恢复

机房 A 故障恢复：

- 恢复节点重新加入集群
- 注意：等机房 A 的 master 数据同步完毕，需进行 leader 切换，确保机房 A Master 还是 leader



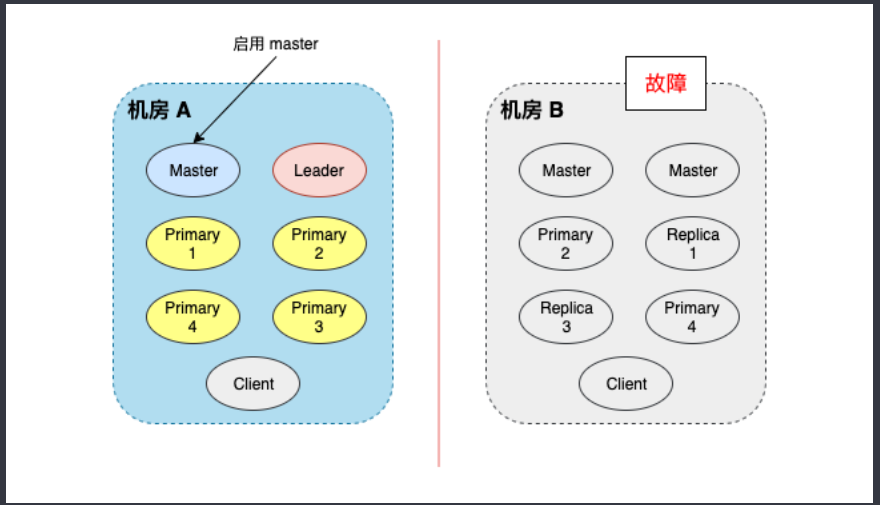
elastic

双机房高可用方案

故障恢复

机房 B 异常：

- 机房 A 中 Master 节点 未过半数
- 但由于 A 机房 Master 为 Leader，故有最新数据
- 为避免集群只读，需启用备用 master
- 备用节点 + Leader 组成过半数 master group



注意：必须确保机房 B master 已不存在才能启用备用节点，以防脑裂！

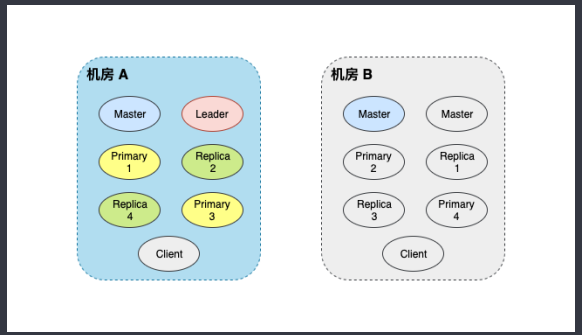
elastic

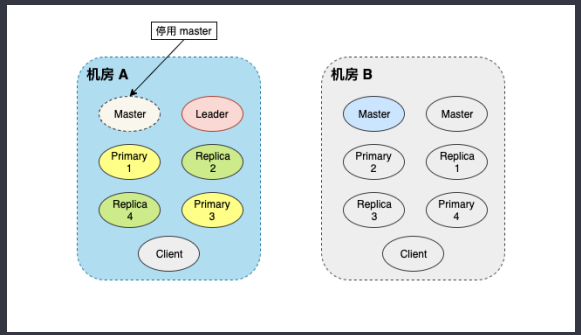
双机房高可用方案

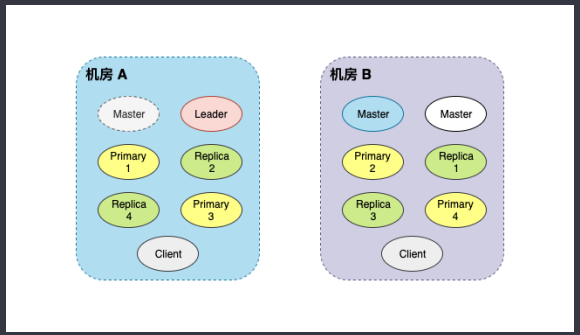
故障恢复

机房 B 故障恢复：

- 启动机房 B 一个 master => 停止备用 master => 启动机房 B 另一个 master







elastic

双机房高可用方案

改进具体实现

控制节点“随机” ID 生成两种方式：

1. 模拟 es 实现
   - 将节点 id 值按 es 存储格式直接写入节点本地数据文件中（node-xxx.st，需要对 es 数据结构十分了解）
2. 控制节点 id 随机生成算法 seed
   - node.id.seed（隐藏选项）

=> 方式 1 侵入性过大，方式 2 扩展性更好

双机房高可用方案

改进具体实现

seed 值（node.id.seed）	根据 es 算法生成的 nodeId
10	0nr9uvgWRXK6Xf9BtKW0aQ
33	1vdeutcQQrWXbePVBhTfCg
92	2nTAubYKT_m0fMdpWIlJrA
123	3fEhuZUETDyRjKv9qvAzTQ
169	4nYfvdydRmyjCnx90vA-6Q
196	5vOAvLuXQ7CAGmARJF9pig

双机房高可用方案

改进具体实现

其他注意事项：

- 第一次启动，先启动候选 leader 节点
- 需对集群状态做持续监控，如发生 leader 漂移需及时修正
- ES >= 7.0 版本修改了选举算法（适配方案还在探索中）

部署优化工具

集群自动化部署方案

目前的自动化部署工具：

- 不易部署多个集群（同一组机器）
- 部署后无法方便管理，比如批量更新集群配置
- 集群扩容、升级管理支持较弱
- 使用较为复杂

部署优化工具

集群自动化部署方案

我们内部的改进方案：

- 基于 Ansible 开发，无需预装部署 client，只需拥有 ssh 权限
- 自动检查和配置 es 安装环境要求（根据官方指定要求 / 配置须有相应权限）
- 自动安装配置 JDK 环境（配置可选）
- 支持同组机器节点下快速部署多个集群，提高机器资源利用率
- 方便的支持节点扩展、集群配置更新
- 支持生产环境 master/data/client 拓扑关系部署、多机房部署
- 极致简单，一个命令完成全部工作

部署优化工具

集群自动化部署方案 – 使用

- 第一步：修改配置项
  - 拷贝或直接修改 example.cfg 配置文件，填写 集群名称、es 版本.. 等信息。
- 第二步：节点 ES_HEAP_SIZE 配置
  - 根据自身环境配置，修改 vars 目录下，节点角色对应名称配置文件的 jvm_heap_size 默认值至合适大小。
- 第三步：一键安装
  - ansible-playbook -i example.cfg setup.yml -v
- 第四步：启动集群
  - 安装完毕后进入 /home/<部署用户>/elk/elasticsearch目录，内含 start_elasticsearch.sh 启动脚本。执行 sh start_elasticsearch.sh 命令启动 es 服务即可。
- 第五步：部署完毕
  - 执行到这里部署就已经完毕

部署优化工具

集群自动化部署方案 – 效果

- 部署后目录结构清晰

```
project-name/
└── cluster-name
    └── elasticsearch
        ├── config
        │   ├── client0
        │   ├── data0_0
        │   └── master1
        ├── data
        │   ├── data0_0
        │   └── master1
        ├── logs
        │   ├── client0
        │   ├── data0_0
        │   └── master1
        └── start_elasticsearch.sh
```

- 开源地址：https://github.com/bluecll/es-easy-setup

总结及未来展望

通用搜索系统

- 安全与权限管理
- 流控
- 多租户隔离
- 管理平台（setting/mapping 可视化管理）
- 分词管理、分词效果预览
- 等。。。

总结及未来展望

未来规划

- 平台容器化
  - Kubernetes + Operator
  - 进一步提升资源利用率和隔离性
- 向量检索
  - 支持海量特征计算和存储
  - 文本相似性 => 语义相似性
  - 为更多的 AI 业务提供服务支撑

Thanks

Q&A

陈亮亮
hzchenliangliang@corp.netease.com