**Course Outcomes**

Upon successful completion of this course, you will be able to:

* Understand the history and trends of the data analytics field via key industry development projects and academic research topics
* Understand the core cluster architecture and database management system technologies used in big data systems
* Study the key trade-offs between scaling of vertically integrated systems versus horizontally sharded systems
* Comprehend reliability and fault-tolerance in distributed operating systems versus traditional operating systems in the context of storage/compute clusters (File Systems & Task Schedulers)
* Study the key trade-offs between NoSQL distributed database management systems versus traditional relational database management systems
* Comprehend availability and consistency in distributed databases versus traditional databases in the context of node consensus/network partitions (Paxos/Raft, CAP Theorem)
* Develop familiarity with the the Apache software platform (Hadoop [HDFS/YARN], Hive, Pig, Spark, etc.) environment for developing and operating big-data model pipelines.
* Develop familiarity with AWS cloud infrastructure (S3, EC2, EMR, etc.) capabilities for designing and architecting big-data system instances.
* Learn from various open access data/articles for research papers and analytical work.
* Build with various open source software/code for development projects and logistical work.

# Module 1: Big Data Concepts

**Readings**

* [Sarker, I.H. Data Science and Analytics: An Overview from Data-Driven Smart Computing, Decision-Making and Applications Perspective. SN COMPUT. SCI. 2, 377 (2021). https://doi.org/10.1007/s42979-021-00765-8](https://doi.org/10.1007/s42979-021-00765-8)
* [H. Hu, Y. Wen, T. -S. Chua and X. Li, "Toward Scalable Systems for Big Data Analytics: A Technology Tutorial," in IEEE Access, vol. 2, pp. 652-687, 2014, doi: 10.1109/ACCESS.2014.2332453.](https://ieeexplore.ieee.org/document/6842585)
* [Acharjya, Debi Prasanna, and Kauser Ahmed. "A survey on big data analytics: challenges, open research issues and tools." *International Journal of Advanced Computer Science and Applications* 7.2 (2016): 511-518.](https://thesai.org/Downloads/Volume7No2/Paper_67-A_Survey_on_Big_Data_Analytics_Challenges.pdf)
* [Lazer D, Kennedy R, King G, Vespignani A. Big data. The parable of Google Flu: traps in big data analysis. Science. 2014 Mar 14;343(6176):1203-5. doi: 10.1126/science.1248506. PMID: 24626916.](https://gking.harvard.edu/files/gking/files/0314policyforumff.pdf)
* [Oussous, A., Benjelloun, F., Ait Lahcen, A., & Belfkih, S. (2018). Big Data technologies: A survey. Journal of King Saud University - Computer and Information Sciences, 30(4), 431-448. https://doi.org/10.1016/j.jksuci.2017.06.001](https://doi.org/10.1016/j.jksuci.2017.06.001)
* [Inoubli, Wissem, et al. "An experimental survey on big data frameworks." Future Generation Computer Systems 86 (2018): 546-564.](https://arxiv.org/pdf/1610.09962.pdf)

## From Data to Value

**第一部分：从数据到价值（From Data to Value - Part 1）**

1. **数据的哲学起点**
   * 引用哲学家 **乔治·贝克莱（George Berkeley）** 的观点：

“存在就是被感知（To be is to be perceived）。”

* + 他认为现实只在被感知时才有意义。类比到数据科学：数据只有在被描述、存储、测量和分析后才对人类或机器有价值。

1. **数据的定义与挑战**
   * 数据不仅仅是数值或测量，它必须带有 **语义（semantics）**，即“意义”。
   * 例如：
     + 98.6 如果不加说明，可能是 **体重（磅）**，也可能是 **体温（华氏度）**。
   * 在数据库中我们用 **元数据（metadata）** 或 **schema** 来表达数据的语义。
   * 但很多复杂数据（图像、视频、医生笔记、法律文件、科研论文等）无法仅靠简单标签说明其意义。
     + 比如一张图片，数据库里可能只是标注为 picture，但实际内容可能是一只猫骑在马背上，这种 **复杂语义** 远非简单标签能涵盖。
   * 当今大多数数据都是 **半结构化（semi-structured）** 或 **非结构化（unstructured）** 的。
2. **数据的价值与作用**
   * 数据被认为是现代社会、商业、医学和科学的 **核心原材料**。
   * 用途包括：
     + 识别模式和趋势（patterns & trends）
     + 进行预测（predictions）
     + 提升企业盈利（customer preference、inventory trend）
   * 类比：数据像 **原油（crude oil）**，未经处理时杂乱无章，但经过清洗与转化后，就能发挥巨大价值。
   * 经验法则：

80% 的数据科学工作在 **数据准备**（收集、清洗、转化、探索），  
20% 用于实际建模和分析。  
甚至有个玩笑说：“数据科学家 80% 的时间在准备数据，20% 的时间在抱怨准备数据。”

1. **数据科学的系统化方法**
   * 数据科学不是随意的，它是一种 **系统化流程**，涵盖：
     + 数学、统计学、人工智能、计算机工程
     + 使用科学方法释放数据的价值
   * 核心是 **数据科学管道（Data Science Pipeline）**：
     + 一系列 **可重复的步骤**，将原始数据转化为洞见并传递给利益相关者。
   * 这个流程通常是 **迭代的**（非线性），会不断返回之前的步骤优化。

**第二部分：数据科学流程（From Data to Value - Part 2）**

1. **数据科学管道的阶段**
   * **业务理解（Business Understanding）**
     + 明确问题：我们要回答什么问题？目标结果是什么？
     + 确定利益相关者：谁会使用结果，他们的需求是什么？
   * **数据收集（Data Collection）**
     + 从内部和外部来源收集相关数据。
   * **数据准备（Data Preparation）**
     + 处理缺失值、异常值、不一致性
     + 整合不同来源的数据形成统一数据集
     + 数据变换：缩放、格式转换、归一化、特征工程
   * **数据探索（Data Exploration）**
     + 使用统计和可视化方法理解数据
     + 提出假设、发现模式
   * **建模（Modeling）**
     + 选择合适的机器学习或统计方法
     + 在准备好的数据集上建立模型
   * **部署与沟通（Deployment & Communication）**
     + 将模型或分析结果整合进业务系统
     + 清晰、简明地向利益相关者解释结果，确保可操作性
   * 注：现代数据科学管道越来越重视 **伦理问题（ethics）**，避免数据分析加剧偏见或歧视。
2. **传统数据科学流程的局限性**
   * 早期依赖于 **单机服务器、集中式数据库、内存计算**，只适合小规模数据。
   * 随着互联网、电子商务、电子健康、社交媒体、物联网传感器的发展，数据量呈爆炸式增长，出现了 **大数据（Big Data）现象**。
   * 限制主要体现在 **五个方面**：
     + - **可扩展性（Scalability）**
         1. 传统工具（Python、scikit-learn、Pandas、R 等）假设数据能装入内存，大数据时代难以扩展。
       - **性能（Performance）**
         1. 单机处理速度慢，硬件升级成本极高，计算能力增长赶不上数据增长。
       - **时效性（Timeliness）**
         1. 传统流程以 **批处理（batch processing）** 为主，难以支持 **实时分析**。
       - **数据利用率（Data Use）**
         1. 常用抽样或数据缩减来降低规模，但可能丢失关键信息。
       - **成本（Cost）**
         1. 传统数据库和存储方案处理海量数据时成本过高。

## Big Data Overview

**1. 大数据不仅仅是“数据很大”**

* 人们直观理解“大数据”就是 **数据量大（Volume）**。
* 但实际上，大数据的概念不仅仅是“规模大”，还包含了它带来的 **巨大挑战（Big Trouble）**。
* 换句话说，大数据是具有一系列复杂特性的海量数据集合，这些特性会让传统的数据科学流程和工具 **难以处理和分析**。

**2. 大数据的相对性**

* 大数据的标准 **不是绝对的**，而是 **相对当前技术水平而言**。
  + 过去：几百 MB 或 GB 就能算“大”。
  + 现在：TB、PB 甚至 EB 才算“大”。
* 随着硬件能力提升（CPU 多核、GHz 频率、32GB 内存、TB 级存储），人们可能以为问题解决了。
* 但实际上，数据增长的速度 **远超硬件发展** → “数据洪流”问题依然存在。

**3. 数据增长的规模**

* **人均数据量**：平均每人 **每秒产生 1.7 MB 数据**。
* **企业数据量**：平均每天 **10 TB 数据**。
* **科技巨头**（Google、Amazon、Facebook 等）的数据规模远超这个平均值。

**4. 大数据来源**

* **日常交流**：语音助手（Alexa、Siri）、对话式 AI（ChatGPT、Gemini、Copilot）。
* **传感器与物联网（IoT）**：智能手表、血糖监测仪、智能手机。
* **智能家电**：智能电视（收集观影习惯）、冰箱、恒温器、门铃摄像头等。
* **科研实验**：
  + 例子：**CERN 大型强子对撞机（LHC）**
    - 每年产生 **30 PB** 数据
    - 实验运行时每秒 **300 GB 数据**，最终需要压缩为 **300 MB/秒的有效数据**
    - 未来的 **高亮度对撞机（HL-LHC，约 2027 年）** → 数据量将扩大 **10 倍**，达到 **EB 级/年**。

**5. 大数据对科学研究的影响**

* 大数据不仅挑战硬件/软件，还在 **颠覆科学研究方法**。
* 传统科学方法：**提出假设 → 数据收集与实验 → 验证假设**。
* 大数据驱动研究：**从海量数据中直接发现模式和规律**，再形成理论。
* Wired 主编 Chris Anderson (2008) 提出：

“理论的终结（The End of Theory）”——科学研究可能不再以假设为起点，而是直接从数据中挖掘规律。

**大数据概述（Big Data Overview – Part 2）**

**1. 大数据的六个核心特征（6V 模型）**

* **Volume（体量）**：数据规模庞大，需要新的存储与计算方案。
* **Variety（多样性）**：结构化、半结构化、非结构化数据并存（表格、JSON、XML、图像、视频、文本）。
* **Velocity（速度）**：数据生成、传输和处理速度极快，要求近实时（real-time）分析能力。
* **Veracity（真实性）**：数据质量、可靠性、正确性至关重要，关系到结论的可信度。
* **Variability（可变性）**：数据格式、属性和标准不断变化（如 CSV、JSON、XML 的差异）。
* **Value（价值）**：从大数据中提取有意义的洞见并非易事，即使在现代技术条件下仍具挑战。

**2. 数据体量的爆炸式增长**

* 人类历史至 2003 年才累计生成 **5 EB 数据**。
* 2013 年起：**每两天** 就能生成 5 EB。
* 预测：到 **2025 年**，全球数据总量将达到 **180 ZB**（泽字节）。

**3. 真实案例与统计**

* **Facebook**：截至 2021 年，每月产生约 **30 PB 数据**。
* **Amazon Prime Day**：
  + 交易量：**3000 亿次**
  + 数据存储：**2000 TB**
  + 数据传输：**800 TB**

**4. 挑战与机遇**

* **存储与管理困难**：传统数据库难以应对。
* **实时价值**：某些大数据若能及时分析，可直接提升 **企业决策和预测能力**。
* **伦理与法律问题**：大规模收集和存储数据带来了 **隐私保护与数据安全** 的新挑战。

## Confounding Factors

**大数据中的混杂因素（Confounding Factors - Part 1）**

在上一部分我们已经提到过：**大数据的体量（Volume）本身就是一个挑战**，因为它需要巨大的存储和计算能力来处理。但事实上，体量并不是最难解决的因素，更关键的是：

* **大数据的多个混杂因素通常是同时出现的**，而不是单独存在的。
* 这就使得处理大数据的难度成倍增加。

**1. Velocity（速度）——大数据的高速度特性**

* **定义**：大数据的速度指的是 **数据生成、收集、传输和处理的速度**。
* **问题核心**：
  + 数据的产生速率通常是 **外部系统决定的**，我们无法控制。
  + 我们能控制的只是：**我们的系统是否有能力承接、存储并及时处理这些数据**，否则就会出现 **数据丢失或延迟分析**。
* **特点**：
  + 数据速度正在快速增加（社交媒体、传感器、机器对机器交互等）。
  + 高速数据流不能忽视 → 因为它往往包含 **实时价值**（实时决策、即时响应）。
* **应用场景**：
  + 点击流（Clickstream）与广告曝光（Ad Impressions）：每秒产生数百万事件。
  + 高频交易（High-Frequency Trading）：需要在微秒级别响应市场变化。
  + 机器对机器（M2M）通信：数十亿设备实时交换数据。
  + 传感器/基础设施监控：实时生成日志数据。
  + 在线游戏：数百万用户并发，每秒多次输入。
  + 医疗/物联网（IoT）：某些应用涉及 **健康与安全**，必须实时分析。
* **类比说明**：
  + **批处理（Batch Processing）**：像从杯子里慢慢喝水 → 数据量和速度可控，不会丢失。
  + **高速数据流（High Velocity Data）**：像从消防水管（fire hose）里喝水 → 数据源不受控，如果系统跟不上，就会丢失数据或延迟洞见。

**2. Variety（多样性）——大数据的数据类型多样性**

* **容易混淆的点**：
  + **Variety ≠ 数据格式**（如 CSV、XML、JSON、JPEG 等），那些属于 **Variability（可变性）**。
* **Variety 的真正含义**：指的是数据的 **组织方式和语义差异**，即提取“意义”的难易程度。
* **三种主要类型**：
  + **结构化数据（Structured Data）**
    - 遵循固定的 **模式/Schema**
    - 通常以 **表格形式（行列）** 存在
    - 每个字段含义明确，如数据库中的客户姓名、ID、交易金额
  + **非结构化数据（Unstructured Data）**
    - 没有预定义结构，语义提取难度大
    - 包括：邮件、语音留言、手写文字、心电图、音频、视频、X 光、MRI 等
    - 通常需要额外处理（自然语言处理、图像识别等）
  + **半结构化数据（Semi-Structured Data）**
    - 介于两者之间，部分有结构，部分无结构
    - 常见于 JSON、XML 等数据格式
* **现实情况**：
  + **约 80% 的商业和科研数据是半结构化或非结构化的**。
  + 因此，如何处理 Variety 是大数据分析的核心挑战之一。

**大数据混杂因素（Confounding Factors - Part 2）**

在上一部分中我们讨论了 **Velocity（速度）** 与 **Variety（多样性）**。在这一部分，重点深入探讨了 **结构化数据、非结构化数据以及半结构化数据** 的区别与挑战。

**1. 结构化数据（Structured Data）**

* **定义**：符合 **固定数据模型（schema）** 的数据，通常以 **表格形式（行-列）** 表达。
* **特点**：
  + 每个字段有 **明确的名称与类型**（如整数、字符串、日期、地址、浮点数等）。
  + 可以添加 **约束条件**（如值的范围、主键/外键关系），保证数据一致性。
* **典型例子**：
  + 关系型数据库（Relational Databases）
  + Excel 电子表格
  + 企业客户信息表、订单记录表

**2. 非结构化数据（Unstructured Data）**

* **定义**：没有固定的结构或预定义模型，内容难以用简单字段标签完全描述。
* **举例说明**：
  + 照片：虽然可以存储在数据库的 BLOB（Binary Large Object）中，并标注为 “picture”，但真正的语义信息（如“这是某种花，花瓣是什么颜色”）无法被简单标签捕捉。
  + 另一张照片可能是“家庭郊游”，即使存有拍摄日期和时间等元数据，主要内容仍难以通过简单字段表达。
* **常见形式**：
  + 图片、视频、音频
  + 流式传感器数据
  + 网页、PDF 文件、PPT 演示文稿
  + 邮件、博客文章、Word 文档
* **现实意义**：
  + 约 **80% 的商业与科研数据是非结构化或半结构化** 的，只有约 **20%** 是标准结构化数据。
  + 非结构化数据在医疗、娱乐、科研、商业中都占据主导地位。

**3. 半结构化数据（Semi-Structured Data）**

* **定义**：同时包含 **结构化部分** 和 **非结构化部分** 的数据形式。
* **复合型特征**：
  + 有清晰的元数据描述，且元数据本身具有独立价值，而不仅仅是附属说明。
  + 内容部分可以承载文本、图像、代码等多样化数据。
* **例子**：
  + **网页（HTML 文件）**
    - Header 部分：结构化的标签和属性（如 <title>, <meta>）。
    - Body 部分：可能包含文本、图片、脚本代码等非结构化内容。
  + **医疗记录**
    - 结构化部分：实验室检测结果（数值、指标名称）。
    - 非结构化部分：医生的主观描述、建议（如“应当减肥”、“建议戒烟”）。
* **模糊性**：
  + 难以精确定义“多少结构化信息”才算半结构化。
  + 一个判定方法：如果结构化部分 **除了描述非结构化部分，还能独立提供价值**，则可认为是半结构化数据。
* **典型格式**：
  + JSON
  + XML
  + HTML
  + 其他灵活组织框架

**4. 总结**

* **结构化数据**：规则严格、含义清晰，便于存储与分析。
* **非结构化数据**：缺乏固定模型，占数据总量的绝大多数，难以直接提取语义。
* **半结构化数据**：兼具结构化与非结构化特点，是现实中最常见的复合型数据形式（特别是在医疗与互联网场景）。

最终目标：  
无论是 **结构化、半结构化还是非结构化数据**，我们的任务都是 **提取有价值的洞见（Insights）**。

**大数据中的混杂因素（Confounding Factors - Part 3）**

在前两部分中，我们讲过 **Volume（体量）、Velocity（速度）、Variety（多样性）**，现在补充大数据的其他几个关键混杂因素：

**1. Veracity（真实性 / 准确性）**

* **定义**：数据必须符合事实，**真实、准确、完整、一致**，我们才能基于它做出有效决策。
* **挑战**：
  + 随着数据体量和速度的增加，人力无法逐条检查其 **偏差、不完整、不准确、不一致**。
  + 需要 **自动化或半自动化** 技术来完成数据真实性的评估与校正。
* **问题示例**：
  + 约会网站的用户填写虚假的身高、体重、职业信息。
  + 即便数据量很大，低质量数据也不会“被淹没”，反而会显著影响结果。
* **影响**：
  + 数据质量差会导致企业 **决策错误、战略无效、运营低效**，甚至造成 **每年数百万美元的损失**。
  + 更严重的后果是：企业或科研组织可能 **声誉受损**，且难以挽回。

**2. Variability（可变性 / 差异性）**

* **定义**：来自不同来源的数据往往存在差异，导致难以整合成统一、可分析的数据集。
* **常见差异维度**：
  + **结构和格式不同**（CSV、JSON、XML、数据库表格等）。
  + **同名字段，含义不同**（如 “ID” 在不同系统中可能指客户ID、交易ID或订单ID）。
  + **数据生成速率不同**（实时流 vs 批处理）。
  + **数据质量不同**（部分来源准确，部分来源噪声大）。
* **应对方法**：
  + 需要构建 **共享的数据本体论（Ontology）或分类体系（Taxonomy）** 来统一语义。
  + 不同来源的数据还要经过清洗、标准化，以保证可比性和一致性。

**3. Value（价值）**

* **定义**：大数据分析的最终目的就是 **挖掘数据中潜在的价值**。
* **矛盾点**：
  + 虽然大数据蕴藏巨大潜力，但要真正释放其价值充满挑战：
    - 数据体量庞大，准备成本高；
    - 数据质量参差不齐；
    - 分析和处理技术复杂；
    - 实际能否转化为 **有用洞见（Insights）和行动（Actionable Decisions）** 仍存在不确定性。
* **现实压力**：
  + 在商业和科研竞争环境下，成功获取数据价值是 **生存与发展的关键**。
  + 这对数据科学家和所使用的技术提出了巨大压力：既要克服大数据的混杂因素，又要在有限时间和资源下交付结果。

## Big Data Challenges and Benefits

A screenshot of a computer

AI-generated content may be incorrect.

**📌 其他挑战（Big Data Challenges）**

* **技术挑战**：传统技术无法高效扩展，实时处理困难。
* **运营挑战**：数据准备工作占用 80% 时间。
* **财务挑战**：数据增长速度远超 IT 预算。
* **伦理与社会挑战**：隐私保护、法规遵循（如医疗/金融数据）、公平与避免偏见（如 Amazon 招聘 AI 案例）。

**🌟 大数据收益（Big Data Benefits）**

* **商业价值**：
  + 理解客户行为 → 个性化产品与服务
  + 提高运营效率 → 消除浪费
  + 风险控制 → 识别欺诈与安全威胁
  + 发现创新机会
* **社会与科研价值**：
  + 医疗：每年节省 **3000 亿美元**，降低 8% 医疗支出
  + 企业：提高利润率超过 **60%**
  + 政府：节省 **1000 亿美元** 运营成本
* **个人/职业价值**（对数据科学家）：
  + 成为组织核心，掌握决策关键权力
  + 提升职业竞争力

## Big Data Technology

**📌 大数据技术 (Big Data Technology)**

**1. 大数据技术 vs 传统数据技术**

* **传统数据技术**（如关系型数据库、ETL 系统）特点：
  + 假设数据能装入内存或单机存储
  + 依赖单个或少数服务器 → **集中式处理（Centralized Processing）**
  + 优点：简单、易于管理
  + 缺点：存储/计算能力有限，单点故障风险大，扩展性差
* **大数据技术**：
  + 专门为应对 **高体量、高速度、多样性** 而设计
  + 运行在 **分布式计算环境（Distributed Processing Environment）**
  + 依赖 **分布式中间件（Distributed Middleware）** 来协调多个节点 → 提供“像一个整体的计算机”的一致性视图

**2. 集中式系统 vs 分布式系统**

| **特性** | **集中式系统 (Centralized)** | **分布式系统 (Distributed)** |
| --- | --- | --- |
| **资源配置** | 单机或少数服务器 | 多节点网络协作（集群或跨地域） |
| **扩展方式** | 垂直扩展 (Vertical Scaling) | 水平扩展 (Horizontal Scaling) |
| **可靠性** | 单点故障，整体崩溃风险大 | 节点冗余 + 容错机制，可靠性更高 |
| **复杂性** | 管理简单 | 管理复杂，需要中间件协调 |
| **成本** | 高性能硬件，成本昂贵 | 使用普通硬件（commodity servers），成本更低 |
| **适用场景** | 中小规模数据处理 | 大数据存储与分析 |

**3. 扩展性（Scalability）**

* **垂直扩展 (Vertical Scaling, Scale Up)**
  + 给单台机器加资源（更多 CPU、内存、存储）
  + 优点：实现简单
  + 缺点：昂贵、性能提升有限、单点风险大
* **水平扩展 (Horizontal Scaling, Scale Out)**
  + 增加更多节点组成集群
  + 优点：弹性强，易于应对大数据
  + 缺点：管理复杂，节点越多故障概率越大 → 需要强大的分布式中间件

**4. 分布式中间件的作用**

* **并行处理**：把任务分配到多个节点，利用并行加速计算
* **资源管理**：协调 CPU、内存、存储、网络的使用
* **容错能力**：通过 **冗余、备份、故障转移 (failover)** 提高系统可靠性
* **透明性**：
  + **扩展透明 (Scaling Transparency)**：应用不需要修改就能扩展节点
  + **位置透明 (Location Transparency)**：不必关心资源的物理位置
  + **协调透明 (Coordination Transparency)**：任务并行执行时避免冲突

一句有趣的描述：

“分布式系统就是：我不能完成工作，因为一台我从没听说过的电脑挂了。”（Butler Lampson）

**5. 云计算与大数据技术**

* 云计算提供：**弹性、按需、低成本** 的计算与存储资源
* 云厂商（AWS、Azure、GCP 等）提供：
  + 预配置的大数据处理服务（即开即用）
  + 云对象存储（Cloud Object Storage）：低成本存储海量结构化/非结构化数据
  + API & GUI 界面操作，快速启用、扩展、关闭服务

因此：**云计算已成为大数据处理的常见环境**。

**6. 典型大数据技术栈**

1. **分布式计算引擎**
   * Apache Hadoop (MapReduce)
   * Apache Spark（内存计算更快）
2. **分布式存储系统**
   * HDFS（Hadoop Distributed File System）
   * NoSQL 数据库（MongoDB, Cassandra, HBase）
3. **云计算支持**
   * AWS EMR / S3
   * Azure HDInsight / Blob Storage
   * Google BigQuery / Cloud Storage
4. **中间件**
   * 任务调度、资源管理、数据分布协调

✅ **一句话总结**：  
大数据技术 = **分布式计算 + 分布式存储 + 容错机制 + 云计算支持**，核心目标是 **应对 6V 挑战，释放数据价值**。

## Generic Distributed Storage Systems and Execution Engines

**一、为什么要分布式？**

* **单机不够用**：容量、内存、带宽、计算力、可靠性都顶不住大数据（体量大、速度快）。
* **目标**：把**海量数据分散存**在很多机器上；把**大任务拆小**并**并行算**，同时保证**容错**与**性能**。

**二、分布式存储：核心构件与机制**

| **角色/机制** | **做什么** | **要点** |
| --- | --- | --- |
| **Data Node（数据节点）** | 存数据“块”（chunk） | 单节点容量有限→把一个超大文件**切块**后分散存放 |
| **Metadata Node（元数据节点）** | 记“文件—块—位置—顺序—权限”等元数据 | 读文件先问它“块在哪儿，顺序如何” |
| **Chunk（数据块）** | 文件被切成连续小段 | 常见固定大小（如 64MB/128MB），便于分布与并行 |
| **Replication（副本）** | 每个块/元数据存多份 | 任一节点挂了仍可读写（高可用） |
| **Checksum（校验）** | 写入与读取时计算并比对 | 发现块损坏时，自动换读其他副本 |
| **Middleware（中间件）** | 调度、恢复、路由、均衡 | 失败检测、重试、重建副本、负载均衡等都靠它 |

小结：**切块 + 位置表 + 多副本 + 校验 + 中间件**，构成可扩展且可靠的分布式文件系统（如 HDFS 的思想）。

**三、分布式执行：并行计算是怎么“跑起来”的？**

以“求一个超大整数文件的**总和**”为例：

1. **任务拆分（Map/Partial）**

* 把“求和”拆成对**每个块**的“局部求和”。
* 在集群中**并发**启动 N 个“局部求和”任务（N≈块数）。

1. **结果汇总（Reduce/Final）**

* 启动一个“最终求和”任务，收集所有“局部和”，再加一次得到**全局和**。

1. **数据本地化（Data Locality）**

* **把代码挪到数据旁边**：让“局部求和”尽量在**存有该块的同一节点**上运行。
* 这样几乎不走网络大流量，只传小结果，**节省带宽**、**加速整体任务**。

1. **容错**

* 某个执行节点挂了？中间件会在**其他副本所在节点**重启该“局部任务”。

小结：**并行 + 本地化 + 小结果汇总 + 容错重试**，是分布式执行引擎（如 MapReduce/Spark 的基本范式）。

**四、与网络/带宽的关系**

* 把**大数据搬来搬去**会被网络拖慢；
* 通过**数据本地化**，把“算子/函数”发到存块的节点上运行，只有小量中间结果跨节点传输；
* 有些工作负载（例如全局排序、连接）**仍需数据洗牌（shuffle）**，这时引擎会做管道化、压缩、聚合、分区等优化来减少网络成本。

# Module 2: Apache Hadoop Overview

**Key concepts**

* Hadoop
* Hadoop Distributed File System (HDFS) Overview
* Using Hadoop Distributed File System
* Cloud Object Storage for Big Data
* Yet Another Resource Negotiator (YARN)

**Readings**

* [*Hadoop Web Site:* Apache Hadoop. (n.d.). Retrieved January 6, 2024](https://hadoop.apache.org/)
* [Apache Hadoop 3.4.0-SNAPSHOT – HDFS Architecture, Retrieved January 7, 2024.](https://apache.github.io/hadoop/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html)
* [K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Incline Village, NV, USA, 2010, pp. 1-10, doi: 10.1109/MSST.2010.5496972.](https://storageconference.us/2010/Papers/MSST/Shvachko.pdf)
* [K. Shvachko, “The exabyte club: LinkedIn’s journey of scaling the Hadoop Distributed File System, ”, 2021, Accessed January 7, 2024](https://engineering.linkedin.com/blog/2021/the-exabyte-club--linkedin-s-journey-of-scaling-the-hadoop-distr)
* [Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). Association for Computing Machinery, New York, NY, USA, 29–43.](https://static.googleusercontent.com/media/research.google.com/en/archive/gfs-sosp2003.pdf)
* [*Apache Hadoop 3.4.0-SNAPSHOT – Overview*, Retrieved January 67 2024.](https://apache.github.io/hadoop/hadoop-project-dist/hadoop-common/FileSystemShell.html)
* [Cloud Object Storage - Amazon S3  - AWS, Amazon Web Services, Inc. Retrieved January 6, 2024.](https://aws.amazon.com/s3/)
* [What is Amazon S3? - Amazon Simple Storage Service, Retrieved January 6, 2024.](https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html)
* [Cloud Data Storage Deep Dive: S3, GCS, and Azure Blob Storage Compared | Airbyte, Retrieved January 6, 2024.](https://airbyte.com/data-engineering-resources/s3-gcs-and-azure-blob-storage-compared)
* [Wankhede, Pallavi, Minaiy Talati, and Rutuja Chinchamalatpure. "Comparative study of cloud platforms-microsoft, azure, google cloud platform, and amazon EC2." J. Res. Eng. Appl. Sci 5.02 (2020): 60-64.](http://www.ijmlc.org/vol8/668-LC0040.pdf)
* [Apache Hadoop 3.4.0-SNAPSHOT – Apache Hadoop YARN, Retrieved January 6, 2024.](https://apache.github.io/hadoop/hadoop-yarn/hadoop-yarn-site/YARN.html)
* [Vavilapalli, Vinod Kumar, et al. "Apache hadoop yarn: Yet another resource negotiator." Proceedings of the 4th annual Symposium on Cloud Computing. 2013.](https://www1.ece.neu.edu/~ningfang/SimPaper/a5-vavilapalli.pdf)

# Module 3: Apache Hadoop MapReduce

**Key concepts**

* The Path to MapReduce
* MapReduce Overview
* Map Reduce Concepts
* MapReduce Examples
* MapReduce Programming
* MapReduce Optimization

**Readings**

* [Sakr, Sherif, Anna Liu, and Ayman G. Fayoumi. "The family of map reduce and large-scale data processing systems." ACM Computing Surveys (CSUR) 46.1 (2013): 1-44.](https://arxiv.org/pdf/1302.2966.pdf)
* [Doulkeridis, Christos, and Kjetil Nørvåg. "A survey of large-scale analytical query processing in MapReduce." The VLDB journal 23 (2014): 355-380.](https://www.cs.princeton.edu/courses/archive/spring16/cos598F/survey-db-mapreduce.pdf)
* [Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107–113. Bottom of Form](https://static.googleusercontent.com/media/research.google.com/en/archive/mapreduce-osdi04.pdf)
* [What is Amazon EMR? - Amazon EMR. (n.d.). Retrieved January 6, 2024.](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-what-is-emr.html)
* [Data-Intensive Text Processing with MapReduce Jimmy Lin and Chris Dyer (University of Maryland) Morgan & Claypool (Synthesis Lectures on Human Language Technologies, edited by Graeme Hirst, volume 7), 2010.](https://lintool.github.io/MapReduceAlgorithms/ed1n/MapReduce-algorithms.pdf)
* [Leskovec, Jure, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive data sets. Cambridge university press, 2020.](http://infolab.stanford.edu/~ullman/mmds/book0n.pdf)
* [Apache Hadoop 3.4.0-SNAPSHOT - MapReduce Tutorial, Retrieved January 6, 2024.](https://apache.github.io/hadoop/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)
* [Yelp, GitHub - Yelp/mrjob: Run MapReduce jobs on Hadoop or Amazon Web Services, GitHub, Accessed January 6, 2024.](https://github.com/Yelp/mrjob)
* [mrjob — mrjob v0.7.4 documentation, Retrieved January 6, 2024.](https://mrjob.readthedocs.io/en/stable/)

# Module 4: Apache Spark (Part 1)

**Key concepts**

* Spark Overview
* Spark Components
* Concepts
* Creating Spark DataFrames
* Defining Spark Schemas

**Readings**

* [Salloum, S., Dautov, R., Chen, X. et al.Big data analytics on Apache Spark. Int J Data Sci Anal 1, 145–164 (2016).  https://doi.org/10.1007/s41060-016-0027-9](https://doi.org/10.1007/s41060-016-0027-9)
* [Armbrust, Michael, et al. "Spark sql: Relational data processing in spark." Proceedings of the 2015 ACM SIGMOD international conference on management of data. 2015.](https://dl.acm.org/doi/pdf/10.1145/2723372.2742797)
* "[*Learning Spark*](https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.pdf)*” pp. 6 – 17, 22-31, 43-54, 58-61*
* [DataFrameReader functions here](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/io.html)

# Module 5: Apache Spark (Part 2)

**Key concepts**

* Transformation – Rows
* Transformations – Columns
* Transformations – Join
* Transformations – Aggregations
* Transformations – Working with Null Values
* Transformations – Spark SQL
* Transformations – Caching
* Actions
* Actions – Writing Data

**Readings**

* Built-in (pyspark.sql.functions) Functions
  + [col](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.col.html#pyspark.sql.functions.col)(col)
  + [concat](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.concat.html#pyspark.sql.functions.concat)(\*cols)
  + [lit](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.lit.html#pyspark.sql.functions.lit)(col)
  + [count](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.count.html#pyspark.sql.functions.count)(col)
  + [max](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.max.html#pyspark.sql.functions.max)(col)
  + [min](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.min.html#pyspark.sql.functions.min)(col)
* DataFrame Function
  + [DataFrame.drop](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.drop.html#pyspark.sql.DataFrame.drop)(\*cols)
  + [DataFrame.select](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.select.html#pyspark.sql.DataFrame.select)(\*cols)
  + [DataFrame.withColumn](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.withColumn.html#pyspark.sql.DataFrame.withColumn)(colName, col)
  + [DataFrame.withColumnRenamed](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.withColumnRenamed.html#pyspark.sql.DataFrame.withColumnRenamed)(existing, new)
  + [DataFrame.join](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.join.html#pyspark.sql.DataFrame.join)(other[, on, how])
  + [DataFrame.corr](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.corr.html%22%20/l%20%22pyspark.sql.DataFrame.corr%22%20/o%20%22pyspark.sql.DataFrame.corr)(col1, col2[, method])
  + [DataFrame.count](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.count.html#pyspark.sql.DataFrame.count)()
  + [DataFrame.cov](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.cov.html#pyspark.sql.DataFrame.cov)(col1, col2)
  + [DataFrame.groupBy](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.groupBy.html#pyspark.sql.DataFrame.groupBy)(\*cols)
  + [DataFrame.summary](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.summary.html#pyspark.sql.DataFrame.summary)(\*statistics)
  + [DataFrame.dropna](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.dropna.html#pyspark.sql.DataFrame.dropna)([how, thresh, subset])
  + [DataFrame.fillna](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.fillna.html#pyspark.sql.DataFrame.fillna)(value[, subset])
  + [DataFrame.createOrReplaceTempView](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.createOrReplaceTempView.html#pyspark.sql.DataFrame.createOrReplaceTempView)(name)
  + [DataFrame.cache](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.cache.html#pyspark.sql.DataFrame.cache)()
  + [DataFrame.unpersist](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.unpersist.html#pyspark.sql.DataFrame.unpersist)([blocking])
  + [DataFrame.collect](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.collect.html#pyspark.sql.DataFrame.collect)()
  + [DataFrame.count](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.count.html#pyspark.sql.DataFrame.count)()
  + [DataFrame.first](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.first.html#pyspark.sql.DataFrame.first)()
  + [DataFrame.head](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.head.html#pyspark.sql.DataFrame.head)([n])
  + [DataFrame.show](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.show.html#pyspark.sql.DataFrame.show)([n, truncate, vertical])
  + [DataFrame.take](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.DataFrame.take.html#pyspark.sql.DataFrame.take)(num)
* Column Function
  + [alias](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.Column.alias.html#pyspark.sql.Column.alias)(\*alias, \*\*kwargs)
  + [cast](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.Column.cast.html#pyspark.sql.Column.cast)(dataType)
  + [Column.eqNullSafe](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.Column.eqNullSafe.html#pyspark.sql.Column.eqNullSafe)(other)
* GroupedData Function
  + [GroupedData.agg](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.agg.html#pyspark.sql.GroupedData.agg)(\*exprs)
  + [GroupedData.avg](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.avg.html#pyspark.sql.GroupedData.avg)(\*cols)
  + [GroupedData.count](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.count.html#pyspark.sql.GroupedData.count)()
  + [GroupedData.max](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.max.html#pyspark.sql.GroupedData.max)(\*cols)
  + [GroupedData.mean](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.mean.html#pyspark.sql.GroupedData.mean)(\*cols)
  + [GroupedData.min](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.min.html#pyspark.sql.GroupedData.min)(\*cols)
  + [GroupedData.sum](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.GroupedData.sum.html#pyspark.sql.GroupedData.sum)(\*cols)
* SparkSession Function
  + ` [SparkSession.sql](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.SparkSession.sql.html#pyspark.sql.SparkSession.sql)(sqlQuery[, args])

# Module 6: Big Data Streaming and Design Patterns

**Key concepts**

* Stream Ingestion and Processing (Part 1)
* Stream Ingestion and Processing (Part 2)
* Analytic Cluster Pattern
* Data Lake Pattern
* Lambda Architecture

**Readings**

* [*Apache Kafka*. (n.d.-b). Apache Kafka. Retrieved February 10, 2024.](https://kafka.apache.org/intro)
* [Apache Kafka. (n.d.-c). Apache Kafka. Retrieved February 10, 2024.](https://kafka.apache.org/documentation/)
* [Narkhede, N., Shapira, G., & Palino, T. (2017). KAFKA: The Definitive Guide: Real-Time Data and Stream Processing At Scale. O’Reilly : 1st ed., 297 pages.](https://www.confluent.io/resources/kafka-the-definitive-guide/)
* [Structured Streaming Programming Guide - SPARK 3.5.0 Documentation. (n.d.). Retrieved February 10, 2024.](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html)
* [R. Hai, C. Koutras, C. Quix and M. Jarke, "Data Lakes: A Survey of Functions and Systems," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 12, pp. 12571-12590, 1 Dec. 2023.](https://ieeexplore.ieee.org/document/10107808)
* [Ravat, Franck and Zhao, Yan Data Lakes: Trends and Perspectives. (2019) In: International Conference on Database and Expert Systems Applications (DEXA 2019), 26 August 2019 - 29 August 2019 (Linz, Austria).](https://oatao.univ-toulouse.fr/25043/1/ravat_25043.pdf)
* [M. Kiran, P. Murphy, I. Monga, J. Dugan and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," 2015 IEEE International Conference on Big Data (Big Data), Santa Clara, CA, USA, 2015, pp. 2785-2792.](https://escholarship.org/content/qt0t36p3hn/qt0t36p3hn.pdf)
* [Martinekuan. (n.d.). Big data architectures - Azure Architecture Center. Microsoft Learn. Retrieved February 10, 2024.](https://learn.microsoft.com/en-us/azure/architecture/databases/guide/big-data-architectures)

# Module 7: NoSQL Database

**Key concepts**

* Using Databases for Big Data Storage
* NoSQL Database Concepts (Part 1)
* NoSQL Database Concepts (Part 2)
* NoSQL Database Classifications (Part 1)
* NoSQL Database Classifications (Part 2)

**Readings**

* [Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. ACM Computing Surveys (CSUR), 51(2), 1-43.](https://www.researchgate.net/profile/Ali-Davoudian-2/publication/324640550_A_survey_on_NoSQL_stores/links/5bbb999392851c7fde3417e2/A-survey-on-NoSQL-stores.pdf)
* [Corbellini, A., Mateos, C., Zunino, A., Godoy, D., & Schiaffino, S. (2017). Persisting big-data: The NoSQL landscape. Information Systems, 63, 1-23.](https://www.cs.helsinki.fi/u/jilu/paper/NoSQL03.pdf)
* [Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. & Gruber, R. E. (2006). Bigtable: A Distributed Storage System for Structured Data. OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, November, 2006 (p./pp. 205--218).](https://storage.googleapis.com/gweb-research2023-media/pubtools/pdf/68a74a85e1662fe02ff3967497f31fda7f32225c.pdf)
* [DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store. SIGOPS Oper. Syst. Rev.](https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf)
* [James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google’s Globally Distributed Database. ACM Trans. Comput. Syst. 31, 3, Article 8 (August 2013), 22 pages.](https://dl.acm.org/doi/pdf/10.1145/2491245)
* [Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 215–226.](https://dl.acm.org/doi/pdf/10.1145/2882903.2903741)
* [Distributed hash table. (2024, February 4). In Wikipedia.](https://en.wikipedia.org/wiki/Distributed_hash_table)
* [Gupta, A., Tyagi, S., Panwar, N., Sachdeva, S., & Saxena, U. (2017, October). NoSQL databases: Critical analysis and comparison. In 2017 International conference on computing and communication technologies for smart nation (IC3TSN) (pp. 293-299). IEEE.](https://www.researchgate.net/profile/Shelly-Sachdeva/publication/323057709_NoSQL_databases_Critical_analysis_and_comparison/links/5c580d23458515a4c7570a34/NoSQL-databases-Critical-analysis-and-comparison.pdf)
* [Sharma, S. (2015). An extended classification and comparison of nosql big data models. arXiv preprint arXiv:1509.08035.](https://arxiv.org/pdf/1509.08035.pdf)
* [*Polyglot Persistence*. March 18, 2024.](https://martinfowler.com/bliki/PolyglotPersistence.html)
* [E. Brewer, "CAP twelve years later: How the "rules" have changed," in Computer, vol. 45, no. 2, pp. 23-29, Feb. 2012.](https://sites.cs.ucsb.edu/~rich/class/cs293b-cloud/papers/brewer-cap.pdf)
* [Kleppmann, M. (2015). A Critique of the CAP Theorem. arXiv preprint arXiv:1509.05393.](https://arxiv.org/pdf/1509.05393.pdf?source=post_page---------------------------)

# Module 8: Key-Value, Wide-Column and Document Stores

**Key concepts**

* Key-Value NoSQL Database Systems: Redis & Memcache (Dynamo)
* Wide-Column NoSQL Database Systems: HBase & Cassandra (BigTable)
* Document NoSQL Database Systems: MongoDB & CouchDB

**Readings**

* DeCandia, Giuseppe, et al. "Dynamo: Amazon's highly available key-value store." *ACM SIGOPS operating systems review* 41.6 (2007): 205-220.
* Chang, Fay, et al. "Bigtable: A distributed storage system for structured data." *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008): 1-26.
* Lakshman, Avinash, and Prashant Malik. "Cassandra: a decentralized structured storage system." *ACM SIGOPS operating systems review* 44.2 (2010): 35-40.
* [The Battle of the NoSQL Databases – Comparing MongoDB and CouchDB, Mani Yangkatisal, Published July 3, 2020](https://severalnines.com/blog/battle-nosql-databases-comparing-mongodb-and-couchdb/)
* [DB-Engines - Document Stores, Retrieved September 10, 2024](https://db-engines.com/en/ranking/document+store)