

NOSQL VS SQL

肖子彤 张倬 谭凌霄

数据库发展史

the History of Databases

数据分类

- 结构化数据
- 非结构化数据
- 半结构化数据

发展阶段

- 一极独霸：SQL
- 异军突起：NoSQL
- 多极并起，各有千秋

研究方法

Research Methods



- mysqlslap
- smack
- Sysbench
- YCSB(The Yahoo! Cloud Serving Benchmark)
- script

环境

- OS : Ubuntu 14.0.2
- CPU : Intel(R) Core(TM) i5-4210M CPU @ 2.40GHZ
- SQL : MySQL-5.6.0
- NoSQL : MongoDB-2.4.9

衡量标准

- 操作执行速度 (时间尺度)
- 数据文件大小 (空间尺度)

结构化数据集实验

Experiment on Structured Datasets

数据集选择

- 使用脚本生成的统一的结构化数据集
- 使用现有实际数据集，（如上海电信某天流量数据集等）

经过比较，选择了以脚本生成的数据集
可以对表的结构进行更细致的定制
实现sql和nosql中数据属性和规模统一

场景

- 场景S1: 100%插入。用来加载测试数据
- 场景S2: 写多读少 90% 更新 10%读
- 场景S3: 混合读写 60%读, 30%插入, 10% 更新
- 场景S4: 读多写少 90% 读, 10% 插入、更新
- 场景S5: 100%读

操作过程

SQL

- 生成sql脚本
- 预热
- 运行mysqlslap调用sql脚本进行检测
- 查看数据文件大小

NOSQL

- 定制workloads
- 预热
- 运行ycsb加载workloads进行检测
- 查看数据文件大小

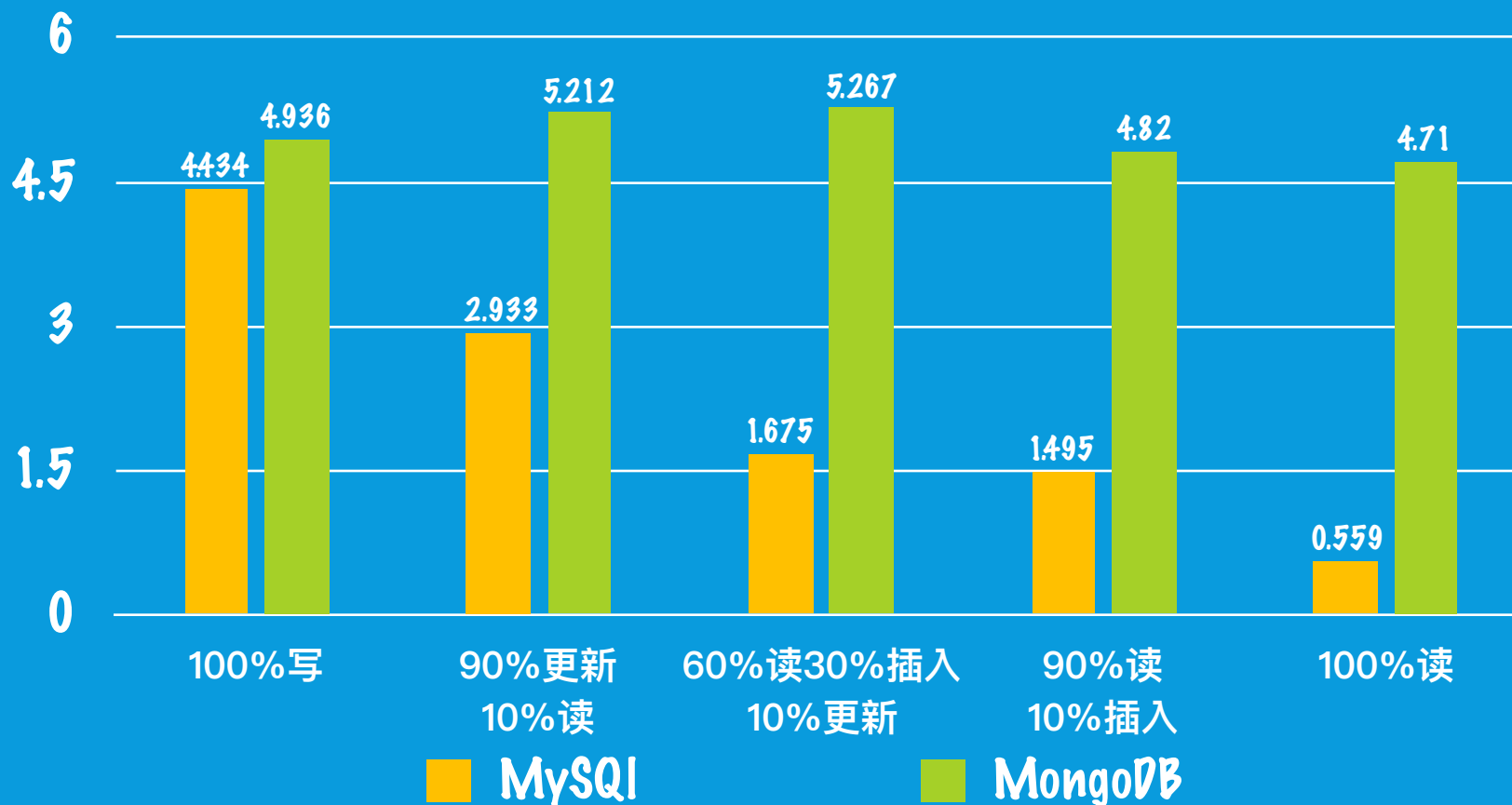
代码片段 (SQL)

```
1 ./gensql.py
+ 18 +-- 6 lines: def genread(totalNum):-----
24
+ 25 +-- 4 lines: def genwrite(totalNum):-----
29
+ 30 +-- 10 lines: def gendelete(totalNum):-----
40
+ 41 +-- 7 lines: def genupdate(totalNum):-----
48
- 49 def gencreate(idNum, eachNum=1):
50     f = open(INSERT_RECORD_FILE, 'w')
51     tmp = []
52     num = 1 / eachNum
53     for i in xrange(idNum):
54         ID = 'user' + os.urandom(9).encode('hex')
55         f.write(ID+'\n')
56         for __ in xrange(eachNum):
57             tmp += ['({}', '{}', '{}', '{}', '{}')'.\
58                     format(ID, os.urandom(50).encode('hex'), os.urandom(50).encode('hex'), os.urandom(50).encode('hex'))]
59         if (i+1) % num == 0:
60             yield 'insert into {}.{} values {};'.format(TEST_DATABASE, TEST_TABLE, ','.join(tmp))
61             tmp = []
62
63 #CONFIG = {gencreate: 1}
64 #CONFIG = {genread: 0.1, genupdate: 0.9}
65 #CONFIG = {genread: 0.6, genupdate: 0.1, genwrite: 0.3}
66 #CONFIG = {genread: 0.9, genupdate: 0.1}
67 #CONFIG = {genread: 1}
68
69
70 if __name__ == '__main__':
71     randbox = []
72     for i in CONFIG:
73         randbox += [i(int(CONFIG[i] * TOTAL_NUM))] * int(CONFIG[i] * TOTAL_NUM + 1)
74     random.shuffle(randbox)
75     #print randbox
76     fout = open(OUTFILE, 'w+')
77     for gen in randbox:
78         try:
79             query = gen.next()
80             fout.write(query)
81         except StopIteration:
82             pass
83     fout.close()
```

代码片段 (INOSQL)

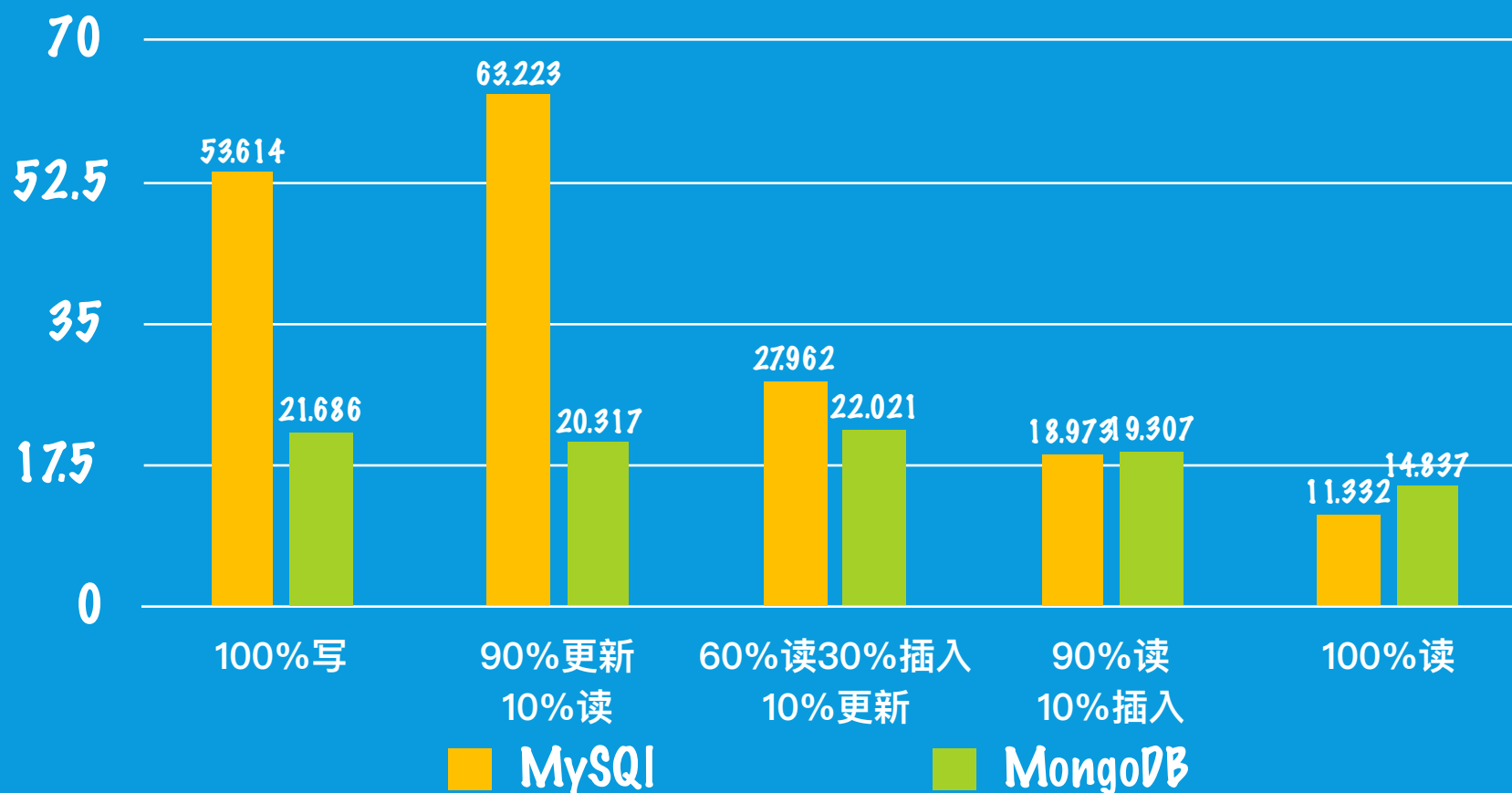
时间性能比较

万级数据时间比较

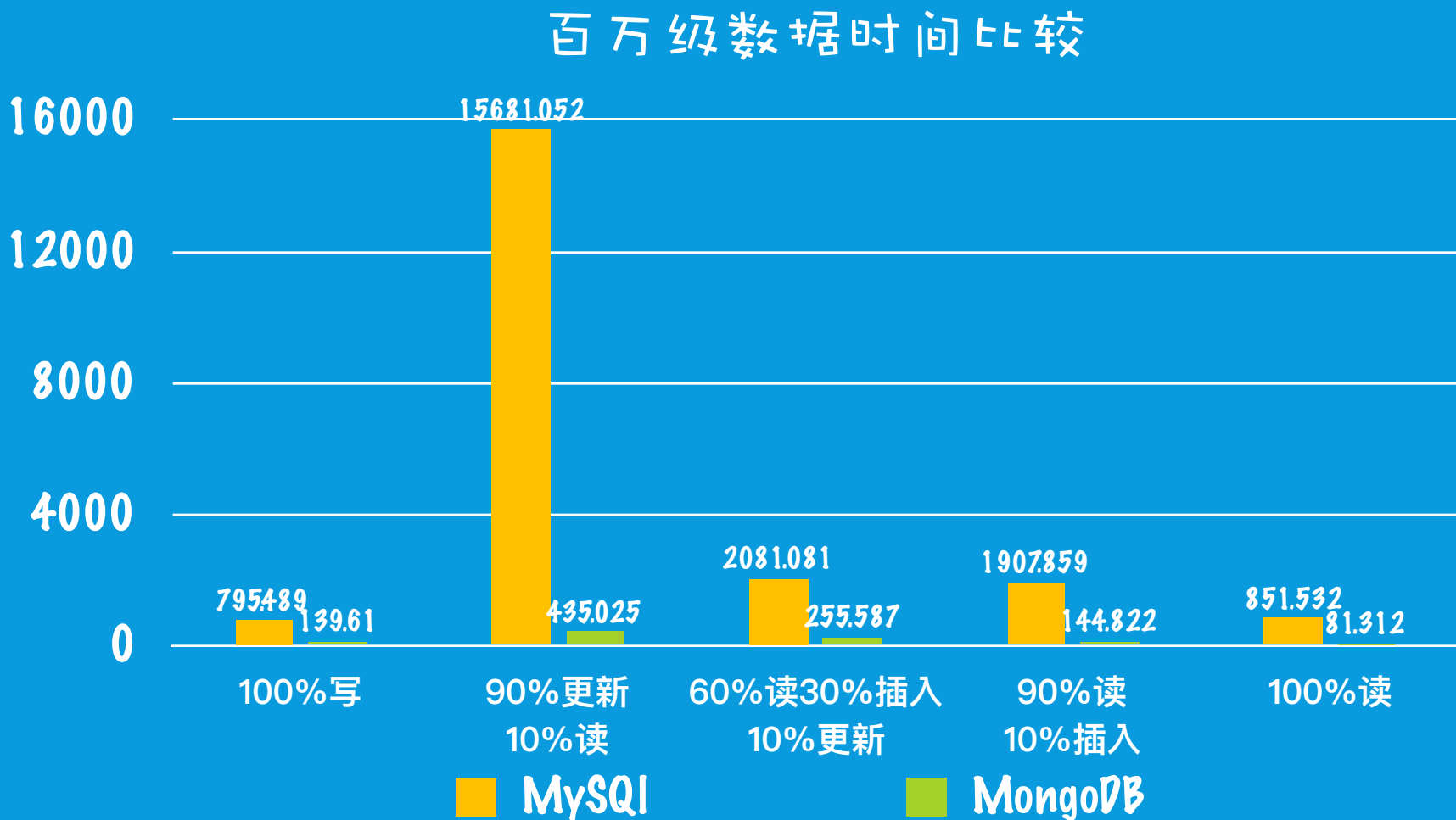


时间性能比较

十万级数据时间比较

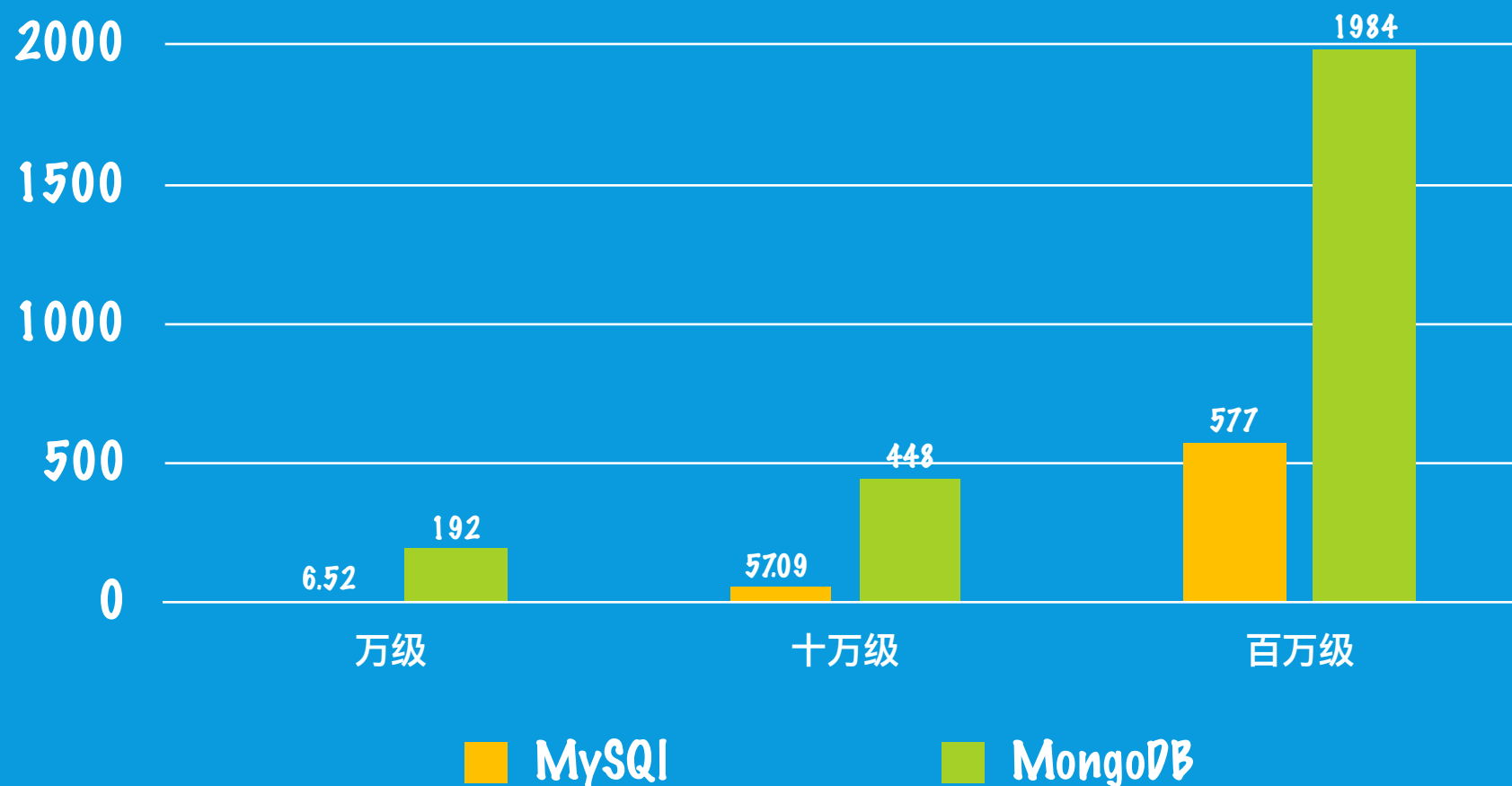


时间性能比较



空间占用

空间占用情况比较



结论

- 当数据规模较小时mysql操作速度略优于mongodb
- 对于大规模数据集MongoDB在操作速度上有极大的优势
- 但其是以巨大的索引空间来获得的这样的优势
- 索引对查询的优化程度能够达到上百倍
- 对于mysql, 更新操作较之其他操作显得尤为耗时
- MongoDB在增删改查等方面消耗的时间近似

非结构化数据集实验

Experiment on Unstructured Datasets

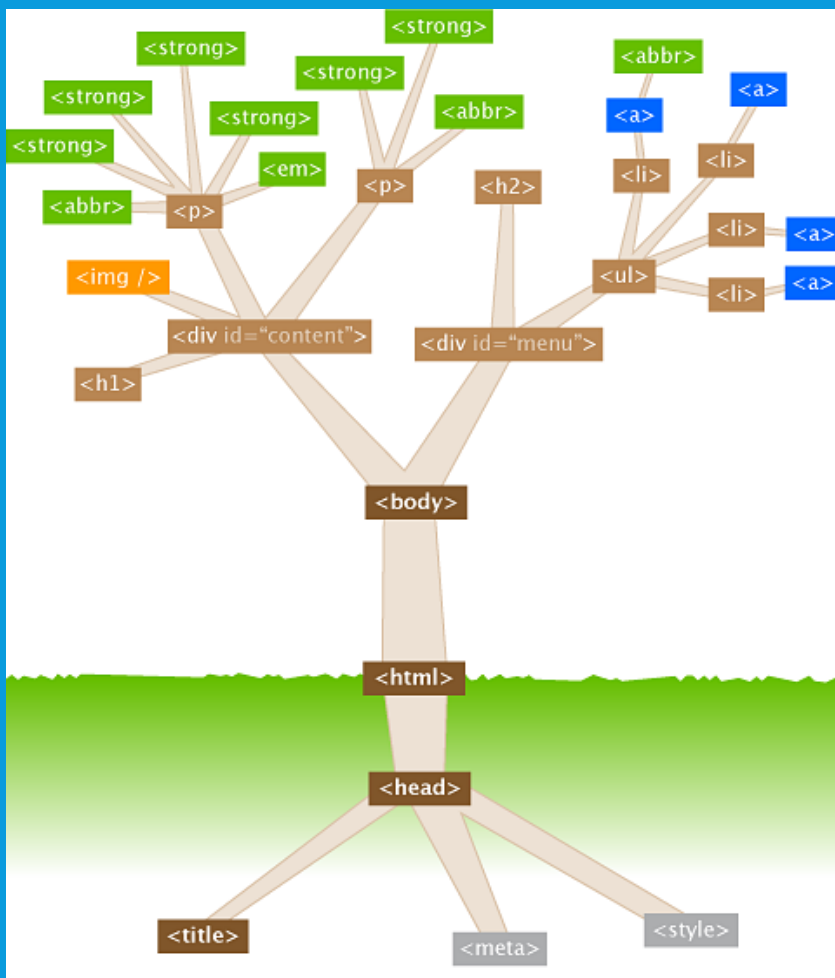
HTML

- 超文本标记语言 (HyperText Markup Language)
- 标签的多重嵌套和属性的延拓性：二维表难以直接表达
- 固定的解析规则和结构：DOM Tree
- 标准的非结构化数据集

HTML

```
<html>
  <head>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge">
    <script charset="utf-8" async="" src="http://xxx.xxx.php?di=1 2 3 4 3 5"></script>
  </head>
  <body>
    <div class="qrcode-wrapper" id="layer" style="display: none">
      <p> It's an example </p>
    </div>
  </body>
</html>
```

DOM TREE



- 文档对象模型（英语：Document Object Model，缩写DOM）
- 将XML（包括HTML）文档解析为由多个节点组成的树形结构
- 元素节点、属性节点、文本节点等（整个xml文档视为一个文档节点）
- 所有的节点和最终的树状结构，都有规范的API，以达到使用编程语言操作文档的目的

爬虫

- 1.将 Start URL 加入解析队列
 - 2.从队列中探出目标URL，通过伪造IP信息和http请求头部，欺骗服务器，获取HTML文本信息
 - 3.解析HTML文本，筛选有用信息，Download至本地
 - 4.提取HTML文本中的链接，形成新的目标URL，判重后加入解析队列
 - 5.队列为空时，结束 Crawling
- 实验中 Start URL 为交大教务处网站首页地址
 - 实验中通过 Scrapy 部署 Python 爬虫

网页解析

SQL

- 解析Html文件为能够被二维表表达的结构 (Get_SQL.py) 存入 MySQL

NOSQL

1. 解析Html文件为Json字符串格式 (Get_NoSQL.py) 存入 MongoDB

Table one: Main

URL	Type	Father	Key (main key)
http://www.baidu.com	#text	1234	1235

Table the: Value

Key (main key)	Value
1235	It%20is%20an%20example.

网页还原，孰优孰劣？

SQL

1. 确定获取网页还原所需信息应执行的sql语句序列（如此复杂以至于需要用脚本来生成这些语句）
2. 依次执行sql语句获取节点信息，进而还原DOM树结构，保存为Json字符串形式（SQL2Json.py）
3. 还原Html文件（Json2Html.py）

NOSQL

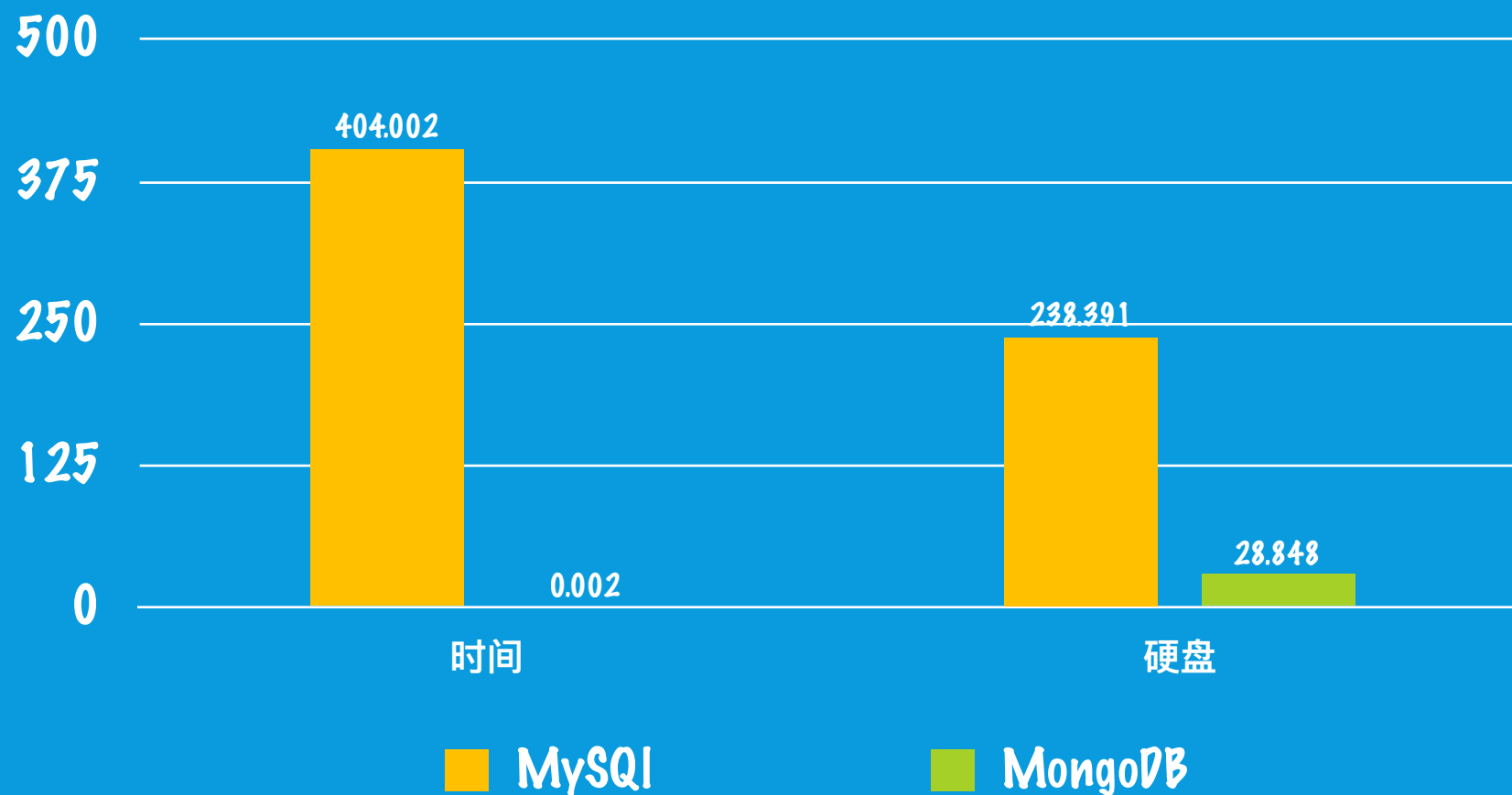
1. 执行一次读操作，获取Json数据
2. 还原Html文件（Json2Html.py）

为保障实验结果的可靠性：Get_SQL.py、Get_NoSQL.py、SQL2Json.py、Json2Html.py 的主要算法部分均由小组成员编写，避免了引入模块造成的性能缺陷

整个实验流程中使用了包括 DFS BFS 表达式解析 字符串匹配等多种算法

结果

网页恢复性能对比



结论

SQL

- 单单查询操作就需几十秒近一分钟，为NoSQL的十多万倍
- 逻辑复杂，不考虑还原过程就已需近百行代码进行辅助工作
- 存储麻烦，需将每个html分解为数千个节点以便存于sql数据库

NOSQL

- 所耗时间为毫秒级
- 逻辑简单，一次查询操作加即可完成
- 存储方便，将Html解析为可存储的Json十分简单

NoSQL在处理非结构化的数据上优势巨大，完虐SQL！

总结

参考文献

- [1]Li Y, Manoharan S. A performance comparison of SQL and NoSQL databases[C]// Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on. IEEE, 2013: 15-19.
- [2]Li X, Zhou W. Performance Comparison of Hive, Impala and Spark SQL[C]//Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2015 7th International Conference on. IEEE, 2015, 1: 418-423.
- [3]Aboutorabi S H, Rezapour M, Moradi M, et al. Performance evaluation of SQL and MongoDB databases for big e-commerce data[C]//Computer Science and Software Engineering (CSSE), 2015 International Symposium on. IEEE, 2015: 1-7.
- [4]Van der Veen J S, Van der Waaij B, Meijer R J. Sensor data storage performance: Sql or nosql, physical or virtual[C]//Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. IEEE, 2012: 431-438.
- [5]Schmid S, Galicz E, Reinhardt W. WMS performance of selected SQL and NoSQL databases[C]// Military Technologies (ICMT), 2015 International Conference on. IEEE, 2015.

GITHUB



- 因为时间有限，我们并没有把我们整个实验展现出来
- 所以我们把相关的材料，程序/脚本的源代码都放到了github上
- 如果大家想重现或者改进我们的项目，可以通过扫描这个二维码查看我们的项目
- 欢迎大家对我们的项目commit

功能展示网页



功能展现网页的二维码

THANKS