实证研究论文阅读笔记

论文详细信息（发表于ESE期刊2018年）：

Experiences and challenges in building a data intensive system for data migration

Scavuzzo, M., Nitto, E.D. & Ardagna, D. Experiences and challenges in building a data intensive system for data migration. *Empir Software Eng* **23,** 52–86 (2018)
doi:10.1007/s10664-017-9503-7
论文地址：https://link.springer.com/article/10.1007/s10664-017-9503-7

选择该论文的原因与动机：

在软件工程领域，数据库是大型软件开发必不可少的一个部分，比如有传统的RDBMS（关系型数据库管理系统）和较新的NoSQL数据库。通过本科的学习，我基本了解了RDBMS的原理和使用方法，但对最近比较流行的NoSQL数据库缺乏了解。因此借这个机会，选择这篇论文，不仅可以更具体的了解实证研究方法，还可以了解NoSQL数据库以及如何进行数据库数据迁移。

一、背景

近年来随着互联网的发展，大数据的作用越来越明显，为了有效地管理大数据，DI(Data Intensive)系统应运而生，它负责管理数据的创建、分析和转换。而DI应用必须有容错性，能容忍硬件、软件、网络问题导致的错误。

在 DI 应用技术之前已经出现了许多大数据处理框架，比如 Hadoop、Spark 和 Flink。这些框架极大地简化了 DI 应用的设计和开发，因为它们提供了任务并行化和调度策略。但是这些框架无法处理一些复杂情况，比如数据模型接口不标

准的问题、如何分配系统资源给这些框架的问题等等，因此需要设计和开发 DI 应用，DI 应用可以提供规则和工具来组合这些框架。

由于当前软件工程领域中没有能够符合上述要求的支持数据迁移的DI应用，所以论文中使用了实证研究的方法来进行设计和开发一个支持数据迁移的DI应用，用开发-测试-重新设计的流程解决问题。在这篇论文里主要研究和改进的DI应用是 Hegira4Cloud，它支持NoSQL数据库中大量数据的离线迁移，而且对准确性、错误容忍度、迁移速度都有很严格的要求。

论文里用到的基础模型 Hegira4Cloud 实际上是作者之前提出过的模型，作者先是在2014年的一篇论文里提出 Hegira4Cloud 的基础模型，这个v1版本主要是完成了不同NoSQL数据库间数据迁移的基本功能。再然后是作者在2016年的另一篇论文里提出了它的改进版本v2，简单地讲解了如何提高Hegira4Cloud模型性能和容错性。而本文的主要贡献是详细地讲解了如何使用反复试验、实证研究的方法来开发Hegira4Cloud模型的v2版本，并验证其可扩展性（可扩展性指的是可以进行大量数据的迁移），最后在理论层面上提出更完善的v3版本（但没有做实验证明）。

二、相关工作

这部分主要讲 NoSQL 数据库、数据迁移、Hegira4Cloud 基础模型这几个相关的工作，以及论文的研究动机。

2.1 NoSQL 数据库

NoSQL 的优点主要是可以处理以下几个 RDBMS（关系型数据库管理系统）

无法高效处理的问题：

1、 数据量大：RDBMS 无法高效存储和检索大量数据，比如查询操作就常常需要处理多个索引、在多个关系表间联合筛选、排序等，而 NoSQL 就没有这个问题。

2、 数据种类多：大量数据常常有不同数据源和格式。

3、 数据处理速度要求严格 ： 大数据处理要求快速写、转换、分析等等。

为了提高可用性，NoSQL 将数据分割到多个数据库节点中，这样用户只需要向包含他们请求的数据的节点查询即可。此外，为了进一步提高可用性和容错能力，NoSQL 还对每个节点进行节点复制/备份，一般存放在不同的数据中心里。由此可见 NoSQL 往往是以分布式系统存在的，因此就需要受到 CAP 原则的约束，即只能在一致性、可用性、分区容错性之间取其二。在必须有分区容错性的前提下，NoSQL 需要在一致性和可用性之间进行权衡。而大多数 NoSQL 数据库倾向于选择可用性，在一致性上进行让步，放弃严格一致性，只需要实现最终一致性。

NoSQL 数据库有多种不同的数据模型，比如 Key-Value, Document, Graph, and Column-family-based 等。在本论文中主要研究的是 Column-family-based NoSQL 数据库。举一个例子了解一下 Column-family-based NoSQL 数据库：

RowKey	ColumnFamily : CF1		ColumnFamily : CF2		TimeStamp
	Column: C11	Column: C12	Column: C21	Column: C22	
“com.google”	“C11 good”	“C12 good”	“C12 bad”	“C12 bad”	T1

以上是一张表，表中只有一条数据，其中 RowKey 即行键，可理解成 MySQL 中的主键列，数据库就是根据 RowKey 来查找数据的；Column 即列，可理解成 MySQL 列；ColumnFamily 即列族，将多个列聚合成一个列族。之所以这样设

计，是因为查询有时不需要将一整行的所有列数据全部返回。列族对应到文件存储结构（不同的 ColumnFamily 会写入不同的文件）；TimeStamp：在每次更新数据时，用以标识一行数据的不同版本。

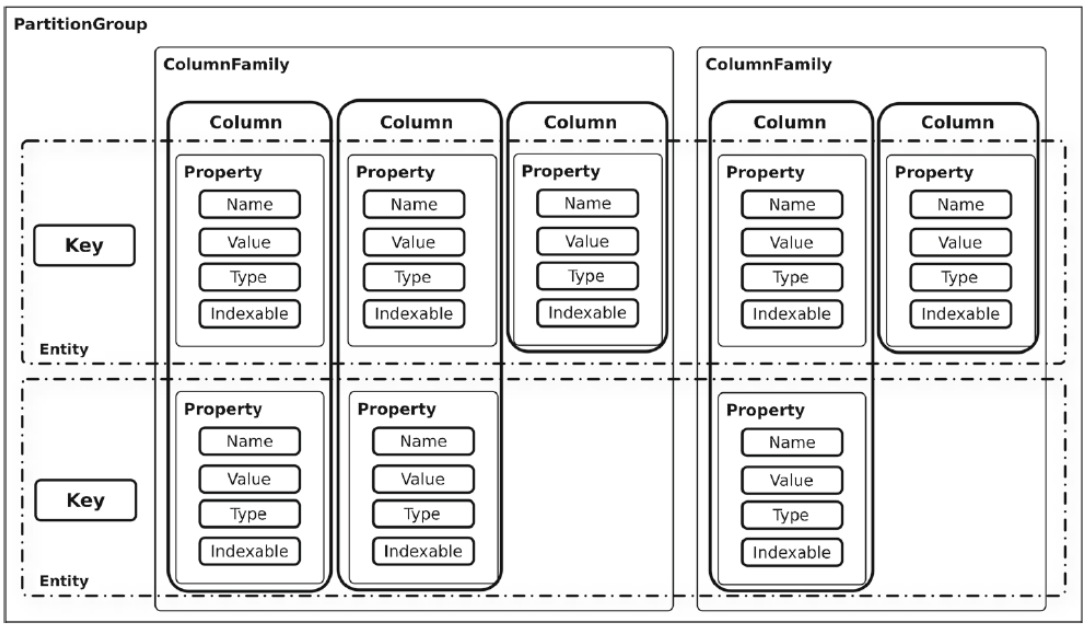
2.2 数据迁移

数据迁移的定义：创建一个源数据库的快照，然后将其传输到目标数据库，并且能够适应目的数据库的技术特性，也就是说这个快照能够在不同种类的 NoSQL 应用间进行迁移。此外，如果数据迁移过程中不允许修改（插入、更新等等），则称为离线（offline）数据迁移。

在 NoSQL 领域中，暂时还没有这样的一种工具，能够以通用的方式来支持在不同类型的 NoSQL 数据库间进行数据迁移。虽然有一些数据库提供了一些工具（比如 Google Bulkloader），以特定的格式（比如 CSV 或者 JSON）进行数据导出和数据导入，但是这些工具只是起到辅助作用，实际上还是需要程序员人工地将数据转换为这种特定的格式才能进行数据迁移，这样的话就称不上一种通用的数据迁移工具。而且这些工具还没有容错机制，因此并不能用于实际的生产环境中。

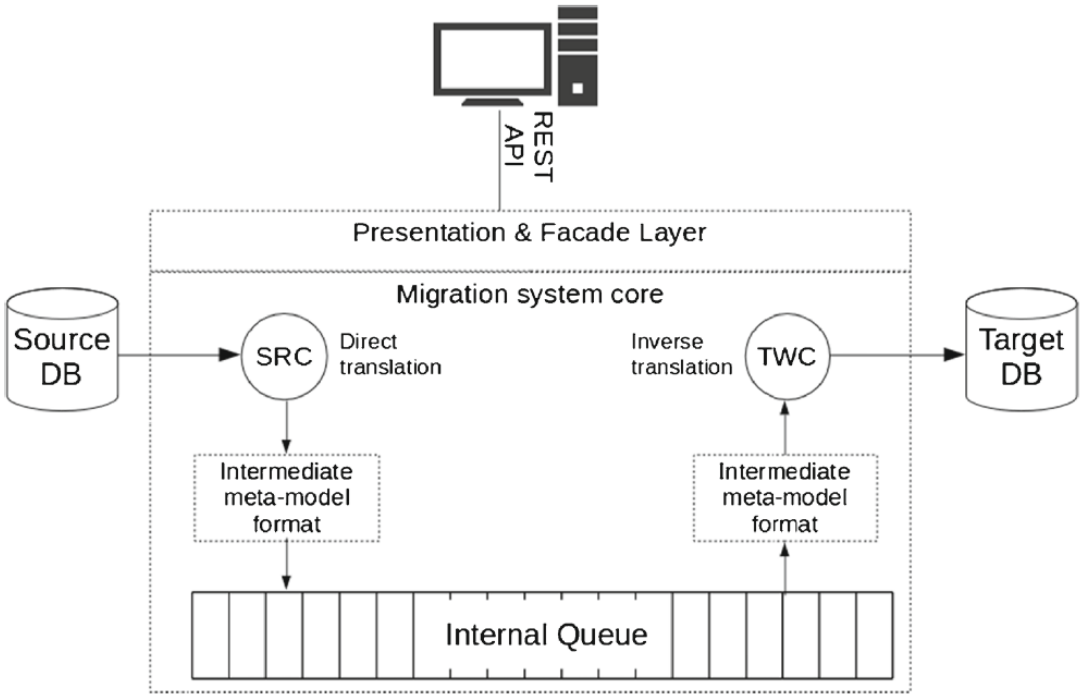
2.3 Hegira4Cloud

Hegira4Cloud 是一个批量处理类型的 DI 应用，适用于 Column-family-based NoSQL 数据库间的离线数据迁移，符合大量、高速、多格式兼容的数据处理要求。Hegira4Cloud 的元模型（meta-model）如下图所示：



可见每个数据实体(entity)有一个唯一对应的 Key, 每个属性(property)由 Name、Value、Type 和 Indexable 组成，一个实体可以是稀疏的，也就是说可以缺失部分属性。有关 Column Family 的部分上面已经 NoSQL 部分章节已讲过，这里不重复。此外还有就是 Column Family 可以归属于相同或者不同的分区（Partition Group）中。这样设计 meta-model 的好处是 Hegira4Cloud 可以保存数据类型、一致性策略和次级索引这三种信息。

Hegira4Cloud 架构图如下：



Source Reading Component (SRC)从源数据库提取数据、转换成 meta-model 模型格式，然后通过 Internal Queue 传输给 Target Writing Component (TWC)，TWC 将其解码成具体数据后将数据存储进目标数据库中。

2.4 研究动机

由上述内容可知，当前 DI 应用领域缺乏一个能有效进行 NoSQL 数据库间数据迁移的工具，虽然有相关的框架和辅助工具，但有许多不足，比如需要程序员适配、缺乏容错机制等等。本论文的研究目的主要是设计和开发这样一款数据迁移的工具，其中基础模型是 Hegira4Cloud 模型，通过实证研究的方法逐步进行改进，即“开发-测试-重新设计”，最后得到满足要求的数据迁移软件。

三、实验

Hegira4Cloud 共有三个版本，v1 版本是作者 2014 年发表的最基础的版本，实现了数据迁移的基本功能，其中还做了两个消融实验，但不是本文重点；v2 版本是作者 2016 年发表的改进版本，但当时只提出理论方法而没有具体的实验研究过程，所以在本论文中用实证研究的方法详细讲解了整个改进过程；v3 版本是本论文提出的新的改进方法，主要针对容错性进行进一步优化，但没有进行实证研究。因此在这一节只讲解 v2 版本的实证研究部分。

3.1 问题定义

确定实证研究问题的定义，实际上就是确定研究问题的范围和研究的具体目标。从上面章节我们知道，v1 版本的 Hegira4Cloud 虽然能完成数据迁移的基本功能，但在其实验过程中不仅出现了几个非法错误，而且性能表现很差（速度慢），显然 v1 版本需要优化。所以本论文的实证研究的问题就是基于 DI 应用 Hegira4Cloud 的 v1 版本进行改进，通过实证研究的方法探索性能瓶颈和改进方法，提升其效果，得到一个能高效完成 NoSQL 数据库间数据迁移功能的软件，具体的目标就是提高数据迁移的速度。

3.2 实证研究规划

本论文的实证研究分以下几步进行：

1、 测试数据库 DaaS 系统的读、写速度上限，判别到底是数据迁移性能已经

接近极限，还是说仍然有较大的改进空间。由于 v1 版本论文的实验结果表明了数据迁移过程中读的速度比写速度快很多，因此即使有读写速度瓶颈，也只可能是写瓶颈而不可能是读瓶颈，所以这里实际上只需要测试写速度的上限即可。由于只测试写速度，只涉及目标数据库，因此只需要以 Azure Table 为目标 NoSQL 数据库进行测试即可。此外还应该设计三种不同大小的数据实体，分别是 754B、1KB 和 4KB，用以判断实体大小对写速度是否有影响。如果仍有改进空间，则继续进行下一步实验。

2、 测试 DaaS 之间的网络带宽是不是数据迁移的性能瓶颈。首先在 Google DaaS 和 Azure DaaS 之间测试，再在两个 Azure DaaS 之间测试。

3、 尝试在 Hegira4Cloud 中加入并行化策略，然后测试是否能获得性能提升。主要是对读数据（SRC）和写数据（TWC）两个环节进行并行化改进。但是由于读速度比写速度快很多，因此可以只用一个 SRT（读线程），而使用多个并行 TWT（写线程）。TWT 的数量可以通过 Hegira4Cloud REST API 设置，所有 TWT 运行在一个线程池中，各自开一个到目标数据库的连接，使用 RabbitMQ 插件和 Round-Robin 算法将 internal queue 中的数据分发给这些 TWT 线程，可以防止因数据分发不均衡导致的线程饥饿的现象。此外应该在 TWT 处添加一个确认机制，防止数据丢失（类似 tcp 的确认机制）。最后为了在发生迁移事务失败时能进行恢复，应该通过 RabbitMQ 将 internal queue 的数据写到磁盘中，这样事务失败后可以直接从磁盘中取出序列化数据继续迁移，而不必从源数据库重新开始。

4、 如果上述步骤获得的性能提升后，仍然有改进空间，则进一步考虑如何优化数据序列化这一部分。V1 版本的序列化方法使用的是 Java 提供的标准

序列化方法，但是这种序列化方法已经被证实存在速度慢、序列数据冗余的问题，因此可以换用其他性能更好的序列化方法。

5、 将以上步骤得到的性能最强的版本投入真实生产环境中进行实验测试，数据集使用一个源自 Twitter 的大型数据集，使用 Hegira4Cloud v2 将数据从 GAE Datastore 迁移到 Azure Table 中，从而验证这个改进版本的 Hegira4Cloud 是否有可扩展性和实用性。

3.3 实证研究执行过程和结果分析

以下实验需要探究如何改进 v1 版本的性能，因此这里放一张 v1 版本实验结果图做对比：

Table 1 Migrations preserving eventual consistency

	From GAE Datastore to Azure Tables			From Azure Tables to GAE Datastore
	dataset #a	dataset #b	dataset #c	dataset #a
Source size (MB)	16	64	512	-
# of Entities	36940	147758	1182062	36940
Migration time (s)	1098	4270	34111	13101
Entities throughput (ent/s)	33.643	34.604	34.653	2.820
Queued data (MB)	81.98	336.73	2709.80	93.50
Extraction&Conversion time (s)	31	120	768	24
Queued data throughput(KB/s)	2707.985	2873.446	3613.067	3989.513
Avg. %CPU usage	4.749	3.947	4.111	0.605

3.3.1 测试读写速度限制

实验执行过程：

在Azure虚拟机上进行实验，其配置为Ubuntu Server 12.04、Microsoft WE data-center，4核CPU、7GB内存。使用log4j library测量实验的持续时间，使用custom library来测量internal queue中的数据量以及Azure Tables中数据实体的

大小。按照实验规划的第一步进行实验，运行三次取平均值得出实验结果。

实验结果：

EAS	ETS(MB)	#threads	Tt (s)	X (ent/s)	X (KB/s)
754 Byte	106	10	217.3	680.0	499.5
754 Byte	106	20	121.3	1218.1	894.8
754 Byte	106	40	102.3	1444.4	1061.0
754 Byte	106	60	122.0	1211.1	889.7
754 Byte	106	192	127.3	1160.7	852.7
1KB	152	10	230.0	642.4	676.7
1KB	152	20	156.0	947.2	997.7
1KB	152	40	129.0	1145.4	1206.6
4KB	580	10	244.0	605.6	2434.1
4KB	580	20	148.0	998.4	4013.0
4KB	580	40	139.0	1063.0	4272.8
4KB	580	60	120.0	1231.3	4949.3
4KB	580	192	116.6	1267.2	5093.7

Entity Average Size (*EAS*)代表数据库中数据实体平均大小;
Entities Total Size (*ETS*)代表数据库的总数据量;
Tt代表所有数据实体写入完毕所需时间;
X代表写速度，有两个度量单位;

结果分析：

可以看到，Azure Tables的写吞吐量在600~1444 entities/s之间。看起来似乎每秒能写的数据实体数是有上限的，但是可以观察到，当数据实体尺寸越大时，以KB/s为度量单位的数据就越大。也就是说，数据库限制了每秒的写操作数，但没有限制每秒的写字节数，可能的原因是数据库为了防止用户请求数过多导致响应时间过长而做出的限制。

此外，在v1版本论文的实验中，实际的数据迁移速度最多只有34.6 entities/s，也就是说写速度不可能是数据迁移的性能瓶颈，同理，比写速度更快的读速度也不可能是性能瓶颈。也就是说Hegira4Cloud是有很大改进的空间的，因此我们继续执行下一步实验。

3.3.2 测试网络带宽

实验执行过程：

一共用到两个Azure虚拟机和一个Google虚拟机。Azure配置与上一个实验相同，而Google虚拟机配置为：Debian 7、Google WE data-center、双核CPU、7.5GB内存。在每个虚拟机中安装iperf工具，用以测量两个虚拟机之间IP网络的带宽。按照实验规划的第二步进行实验，总共两个实验，各自运行五次后取平均值得到实验结果。

实验结果：

首先v1版本的internal queue网络传输吞吐量为24.3Mb/s。

Azure VM与Google VM之间的带宽测试结果为418Mb/s。

两个Azure VM 之间的带宽测试结果为712Mb/s。

结果分析：

由实验结果可见，无论是不同DaaS间还是相同DaaS间的带宽，都远比的internal queue网络传输带宽要大，因此可以确定的说DaaS间的网络带宽不是数据迁移的性能瓶颈。因此继续进行下一步实验。

3.3.3 数据迁移并行化

实验执行过程：

Azure VM的配置与上面实验相同，安装SRC、TWC和一个RabbitMQ插件，测试从GAE Datastore到Azure Tables间的数据迁移。使用log4j library计测量实验的持续时间，使用custom library来测量internal queue中的数据量。使用sysstat package测量CPU使用率。接下来进行实验，将总共147,758个实体、64MB的数

据从GAE Datastore中迁移到Azure Tables去。实验重复进行3次，取平均值作为实验结果。

实验结果：

Table 4 Data Migration from GAE Datastore to Azure Tables

#TWTs	MT(s)	STDM	EET(ent/s)	EDT(KB/s)	ECT(s)	QDT(KB/s)	CPU
10	426	7.37	346.85	254.80	117	2947.11	24.10 %
20	321	5.25	460.30	338.14	158	2182.35	33.06 %
40	306	4.79	482.87	354.72	173	1993.13	35.56 %
60	304	3.81	486.05	357.05	176	1959.16	36.23 %

TWTs代表所用写线程数；
MT代表数据迁移过程所耗费的总时间；
EET和ECT是数据迁移速度的两个度量方法；
CPU代表CPU使用率；
...其他指标略。

结果分析：

从实验结果可知，通过写操作的并行化，最多可以将数据迁移总时间降低至十四分之一，代价是CPU占用提高至单线程的9倍。同时也可以看到，60个TWT线程其实已经达到性能提高的上限，因为60个TWT的效果相比40个TWT的提升已经很小了。当然这个线程数的阈值因硬件性能而异。

此外，我们还可以认为Hegira4Cloud在并行化之后性能仍然有提升空间，因为虽然并行化大幅提升了迁移速度，最高达到了486entities/s，但是对比实验步骤一中的1444 entities/s，还有很大的差距，也就是说需要进一步的优化。然而实验步骤二已经证明了网络带宽不是限制因素，那剩下能优化的就只有数据序列化和反序列化这个部分了。接下来的实验会探究如何改进序列化和反序列化方法。

3.3.4 数据序列化改进

实验执行过程：

我们选用Apache Thrift序列化机制来替换原来的默认序列化方法，Thrift是Facebook开发和开源的一个项目，支持多种复杂的数据类型的序列化。实验配置保持与上一个实验相同，仅改变序列化方法，进行实验。

实验结果：

Table 5 Data migration from GAE Datastore to Azure Tables using Apache Thrift

#TWTs	MT (s)	STDM	EET(ent/s)	EDT(KB/s)	CPU
10	294	3.41	502.58	222.91	58.09 %
20	237	3.87	623.45	276.52	74.66 %
40	168	2.17	879.51	390.10	74.96 %
60	169	0.71	874.31	387.79	75.29 %

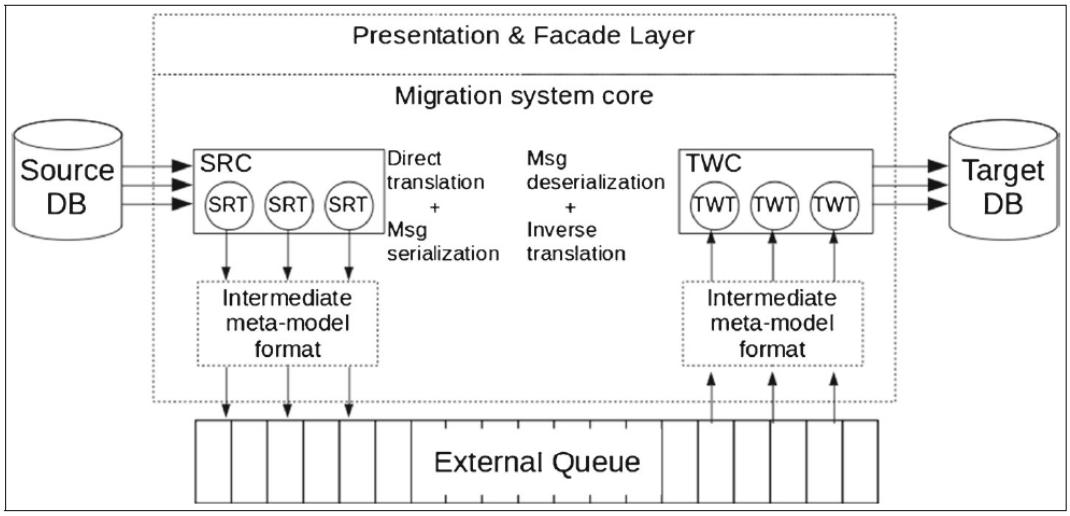
结果分析：

由上述实验结果可见，数据迁移速度已经提升到 879entities/s 了，相当于修改序列化方法之前的 1.8 倍，相当于 v1 版本的 25.4 倍。CPU 使用率之所以大幅提升，主要是因为要进行 Thrift 的 初始化，在启动 120s 后 CPU 使用率就会下降回去，这里之所以保持 CPU 高使用率，是因为数据量太少，很快就迁移完毕了。至此，迁移速度仍然比理论最高值低，理论最高速度大约是当前迁移速度的 1.6 倍，但这属于合理的范围，因为数据迁移过程中牵涉到数据类型、二次索引、一致性策略等信息的保存，所以实际迁移时比理论速度低是正常的。所以可以认为，到此为止已经达到了预期的较好的改进效果，这个改进版本就是 Hegira4Cloud v2 。接下来的实验就是将这个 v2 版本应用到大数据环境下进行测试了。

3.3.5 大数据集测试

实验执行过程：

改进后的Hegira4Cloud v2的架构图如下所示：

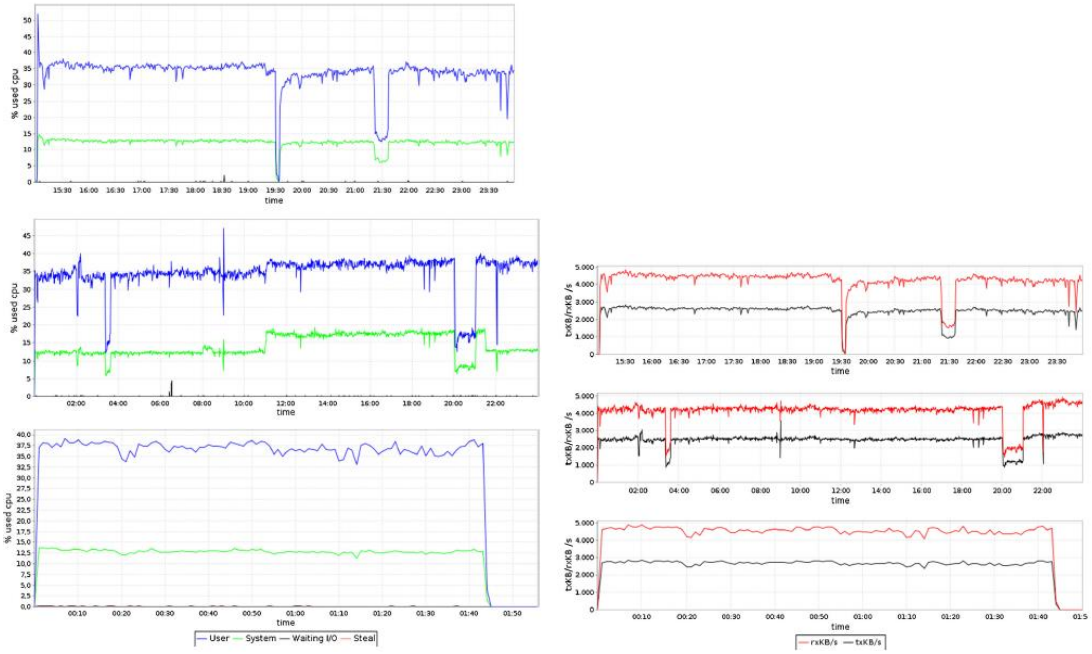


(b) Hegira4Cloud v2

实验数据集包含106,937,997条推特，每条推特除了最长140个字外，还包括用户信息、发表日期、地理位置等详细信息。每条推特作为一个数据实体存储进GAE Datastore中，不同属性的数据类型不同。每条推特的数据实体的平均大小为3.05KB，总共311GB。并行化方面我们使用40个TWT线程，实验环境和配置与上一个实验相同。

实验结果：

数据迁移所用的总时间为34.5个小时，虚拟机的CPU使用率和Hegira4Cloud带宽在这34.5个小时间的变化情况折线图如下所示：



统计所得CPU平均使用率为49.87%。总的迁移速度为856.41entities/s和2,611.63 KB/s。

结果分析：

Hegira4Cloud的带宽十分平稳，除了部分时间点突然下降和回升之外，数据传输速度几乎是相同的。与章节3.3.4的实验（小数据集）相比，每秒迁移的数据实体数几乎相同，但是每秒传输的KB数却是前者的6.69倍，这是因为每条推特对应的数据实体更大，在每秒传输实体数相同的情况下，自然会传输更多的KB数，此实验结果类似于3.3.1章节的实验现象。总的来说，此实验证明了改进版本Hegira4Cloud v2可以应用到大数据集上，且迁移速度非常快。即使无法达到理论最大值，也是因为维持数据特性所必须牺牲的性能。

但是 Hegira4Cloud v2 仍有不足之处。这个大数据集实验进行了三次，其中只有一次是成功的，其他两次都是因为 Google Datastore 发生非法错误（HTTP 500）而失败，错误发生时 Google Datastore 突然停止读操作，并且中断数据迁移过程（防止数据不一致）。也就是说，虽然 v2 设计了写操作部分的数据恢复方法，但

是没有设计如何恢复中断的读操作。这一部分，将在 v3 版本进行改进（v3 版本只有简单讲解而没有实验证明，因此这里省略）。

四、论文贡献与结论

总的来说，本论文展示了如何设计和开发一个 DI 应用，用以支持 column-family-based 类型的 NoSQL 数据库间的数据迁移。本文所开发的 DI 应用的基础模型为 Hegira4Cloud，论文主要讲解了 Hegira4Cloud 的基础架构和运行原理（v1 版本），然后用实证研究的方法对其进行改进，得到 v2 版本，最后引出进一步的更完善的 v3 版本。其中最主要的部分是开发 v2 版本的实证研究过程，通过开发-测试-重新设计的流程进行反复实验，最终得到性能大幅提升的 Hegira4Cloud v2 版本。实验证明了 Hegira4Cloud 可以应用在大型数据集迁移场景下，且性能较高。

五、个人感想与心得

此前在其他课程的学习中，主要都是对软件理论部分进行理解和运用，在这门课程中则是相反，主要关注的是如何应用和实践实证研究方法。通过阅读这篇论文以及按照实证研究的基本概念、技术和过程方法总结论文阅读笔记，我对软件工程领域的实证研究方法有更深刻的理解，包括如何定义研究问题的范围和具体目标 、如何做实验规划、执行和结果分析。我不仅了解了实证软件工程的一个具体的应用例子，还了解了软件工程DI应用领域的最新进展。

此外，我还了解到了NoSQL的一些特性，具体研究了column-family-based类型的NoSQL数据库，以及数据库间数据迁移的流程和用到的技术。论文中先探索性能瓶颈，再执行改进方法的实验规划，给了我很大的启示，即先确定软件是否有改进空间，再进行设计、开发和测试 ，而不是直接设计如何改进然后进行开发，因为如果软件某个部分存在性能瓶颈，则其他部分再怎么改进也没有意义，只是浪费时间和人力而已。

我认为这篇论文写得很好，实验规划合理、实验过程描述详细。但是我觉得还有不足之处，比如说大数据集测试部分的实验，总共311GB的数据实际上并不算很多数据，整个数据迁移过程就已经用到30多个小时了，大型互联网企业的数据库远不止这么多数据，这样看来这个数据迁移应用应该只适用于小型软件的数据库。此外，论文只对v2版本进行实证研究，详细描述了v2版本的开发、改进过程，却对最终版本的v3部分只是稍微一提，没有进一步的实验证明，最终只是得到一个半成品的v2版本。可以看出这也是论文作者的一个套路，他所发表的有关Hegira4Cloud的论文，都是每篇先用实证研究方法详细讲解上一篇论文提出的版本，然后再提出一个新的版本，而这个新版本要等到两年后新论文发表才进行实证研究，吊人胃口。

总的来说，结合实证研究的知识来阅读这篇论文之后，我更清晰和具体的明白了实证研究的概念和流程，也了解了NoSQL数据库数据迁移的最新进展。