Novel large-scale data analytics engines such as Shark Spark [35], Dremel [19], F1 [30], Myria [15] and others [3, 32] use massive parallelism in order to support complex queries on large data sets. These engines are designed to evaluate queries in main memory, because they use sufficiently many servers to ensure that their data fits in main memory. For these engines, the tradi tional performance metrics consisting of the number of disk I/Os is replaced by a new bottleneck, which is the communication cost for reshuffling data during query execution. Each reshuffling step requires a repartition of the entire dataset or intermediate result, which can be expensive. Data shuffling can also create load imbal ance (a.k.a skew) between operator partitions.

Shark Spark [35]、Dremel [19]、F1 [30]、Myria [15]等[3， 32]等新型大规模数据分析引擎使用大规模并行来支持对大型数据集的复杂查询。这些引擎旨在评估主内存中的查询，因为它们使用足够多的服务器来确保其数据适合主内存。对于这些引擎，由磁盘 I/O 数量组成的传统性能指标被一个新的瓶颈所取代，即在查询执行期间重新洗牌数据的通信成本。每个重新洗牌步骤都需要对整个数据集或中间结果进行重新分区，这可能很昂贵。数据洗牌还可以在操作员分区之间创建负载不平衡（也称为倾斜）。

The workloads on traditional OLAP engines usually consist of star-joins with aggregates, where the fact table is significantly larger than the dimension tables. These queries are often optimized by partitioning the fact table and replicating the dimension tables on all workers. But new data analytics engines face new kinds of workloads, where multiple large tables are joined, or where the query graph has cycles. For example, Yaveroˇ glu et al. [37] have recently discovered that the structure of a complex network can be characterized by counting various patterns in the graph. Each pat tern, called a graphlet, represents a small graph. It could be, for example, a triangle, or the complete graph 5. The frequencies of graphlets in the network represent an important statistics for ana lyzing the network’s structure.

传统 OLAP 引擎上的工作负载通常由具有聚合的星型联接组成，其中事实表明显大于维度表。这些查询通常通过对事实数据表进行分区并在所有工作线程上复制维度表来优化。但是，新的数据分析引擎面临着新的工作负载类型，其中连接了多个大型表，或者查询图具有周期。例如，Yaveroˇ glu等[37]最近发现，复杂网络的结构可以通过计算图中的各种模式来表征。每个 pattern，称为 graphlet，代表一个小图形。例如，它可以是一个三角形，也可以是完整的图形 5。网络中图形的频率代表了分析网络结构的重要统计数据。

However, computing these patterns is computationally expensive. Most graphlets have cycles, and in volve 5-10 self-joins on the network data.

计算成本高昂。大多数图形都有循环，并且网络数据上有 5-10 个自连接。

In this paper, we describe a system that can compute efficiently complex join queries, including queries with cyclic joins, on a mas sively parallel architecture. We build on two lines of work that in troduce a novel parallel [5, 8, 9], and a novel sequential [23, 33] algorithm respectively. While the former has been studied only theoretically, the latter is in use in the LogicBlox DBMS. Our first contribution is to empirically evaluate these two recent algorithms together, compared to the traditional ones, explaining when and why they are better. Then we describe two new key contributions that allow the parallel and the sequential algorithm to be deployed efficiently in parallel systems.

在本文中，我们描述了一个系统，该系统可以在一个完全并行的架构上有效地计算复杂的连接查询，包括具有循环连接的查询。我们建立在两条工作线上，分别引入了一种新的并行算法[5， 8， 9]和一种新的顺序算法[23， 33]。虽然前者仅在理论上进行了研究，但后者已在 LogicBlox DBMS 中使用。我们的第一个贡献是将这两种最新的算法与传统算法进行比较，并解释它们何时以及为什么更好。然后，我们描述了两个新的关键贡献，它们允许在并行系统中有效地部署并行和顺序算法。

All traditional engines1 compute conjunctive queries using a tree of join operators. It is well known that, if a query is cyclic, then query plans can be highly suboptimal, no matter what join order one chooses. If one computes the query ( ) ( ( ) ), which lists all triangles, as a sequence of two join opera tors, then the size of the intermediate join is much larger than that of the final answer, because there are typically many more paths of length two than triangles. This was not considered to be a ma jor issue in traditional engines, because cyclic queries were rare. But modern data analytics engines must support such queries fre quently, and they require new approaches.

所有传统引擎1 都使用联接运算符树来计算联合查询。众所周知，如果查询是循环的，那么无论选择何种联接顺序，查询计划都可能非常不理想。如果将列出所有三角形的查询 （ ） （ （ ） ） 计算为两个连接操作的序列，则中间连接的大小比最终答案的大小大得多，因为长度为 2 的路径通常比三角形多得多。在传统引擎中，这并不被认为是一个问题，因为循环查询很少见。但是，现代数据分析引擎必须经常支持此类查询，并且需要新的方法。

Recently, Ngo et al. [23] and Veldhuizen [33] have described novel sequential algorithms that compute a query with multiple joins in one shot, avoiding the computation of intermediate results. These are sequential algorithms, and their runtime has been proven to be worst-case optimal, meaning that it is bounded by the largest possible output that the query can produce for inputs of a given size. Both algorithms require the data to be preprocessed. For par allel computation, Afrati and Ullman [5] have described an algo rithm that computes any multi-join query in a single communica tion round. Beame at al [8, 9] refined this algorithm, calling it HyperCube, and performed a theoretical analysis proving that it is optimal. However, the optimality criterion described by Beame et. al. is not practical, because it assumes that the available servers can be partitioned into sets with fractional number of servers.

最近，Ngo等[23]和Veldhuizen[33]描述了一种新的顺序算法，可以一次性计算具有多个连接的查询，避免了中间结果的计算。这些是顺序算法，它们的运行时间已被证明是最坏情况下的最优，这意味着它受查询可以为给定大小的输入生成的最大可能输出的限制。这两种算法都需要对数据进行预处理。对于等位计算，Afrati 和 Ullman [5] 描述了一种算法，用于计算单个通信轮中的任何多联接查询。Beame 在 al[8， 9] 中改进了该算法，将其称为 HyperCube，并进行了理论分析，证明它是最优的。然而，Beame 等人描述的最优性准则。是不切实际的，因为它假定可以将可用的服务器分区为具有小数个服务器的集合。

Westart by performing an empirical evaluation of the above new sequential and parallel algorithms and compare them to standard reshuffling and join computation methods. In experiments on the Twitter and Freebase datasets, we find that conjunctive queries with large intermediate results can execute up to 8x faster when using a HyperCube shuffle than a traditional one dimensional shuffle. They also transmit up to 98 percent less data.

我们首先对上述新的顺序和并行算法进行实证评估，并将其与标准的重洗和连接计算方法进行比较。在 Twitter 和 Freebase 数据集上的实验中，我们发现，使用 HyperCube 随机排序时，具有大型中间结果的联合查询的执行速度比传统的一维随机排序快 8 倍。它们传输的数据也减少了 98%。

Furthermore, Tributary join, our implementation of LFTJ based on sorted relations, further cuts runtimes by up to 80 percent and CPU times by up to 71 percent，when used in conjunction with a HyperCube shuffle.

此外，我们基于排序关系实现的 Tributary join 进一步缩短了 80% 的运行时间和 71% 的 CPU 时间，当使用时与HyperCubeshuffle结合使用。

Wethen considerpracticalaspectsofbothalgorithms. FortheHyperCube algorithm,wedesignanewapproachtooptimizethenumberof serversharesper variable,which always results in an integral number of shares, thus overcoming a key limitation of prior work[8, 9].

然后，我们考虑两种算法的实际方面。

对于HyperCube算法，我们设计了一种新的方法来优化每个变量的服务器份额数，该变量总是产生一个完整的份额数，从而克服了先前工作的关键局限性[8,9]。.我们以实证方式证明其性能;例如，我们发现，与现有理论方法的幼稚应用相比，它将每个工作线程的最大数据量减少了一半（图 11），并且其每个服务器的工作负载永远不会大于理论上最优负载的 1.06 倍。

对于顺序多联接，我们描述了Tributaryjoin，我们实现了LFTJ的API[33]。LogicBlox 的 LFTJ 实现将每个数据库关系存储在 B 树中。在设置中，数据预处理不是可能的，因为多联接是在洗牌步骤之后执行的;取而代之的是，Tributaryjoins简单地对关系进行排序并操作数组，而不是B树。然后，描述了一种新的变量阶选择优化方法，并进行了实证验证;例如，我们表明，与未优化的运算符执行相比，我们的 Tributaryjoin 优化算法可以将运行时间缩短一个数量级（表 2）。

总而言之，我们做出了以下贡献：、

1.我们对HyperCube洗牌和支流连接一起进行了实证评估（Sec.3）。

2.提出了一种实用的算法来计算HyperCube算法的最优配置（Sec.4）。

3.介绍了一种新的优化算法，用于选择支流连接（Sec.5）的变量顺序。

在本节中，我们介绍了高效连接查询评估方法背后的两个理论构建块：

超立方体洗牌[5]和一个新的顺序多路连接算子，我们称之为Tributaryjoin，我们在LFTJ[33]中实现了API。

我们的目标是在分布式架构上计算查询，使用通过网络连接的服务器。我们假设数据最初是在服务器上均匀分区的，例如使用哈希函数或轮询机制，我们的目标是平衡服务器之间的计算均匀。传统的查询计算方式是首先构造一个查询计划，由几个连接组成，然后通过计算一个连接来评估查询，使用基于哈希的算法。这需要至少等于查询树深度的多个通信轮次 （joinsondifferentbranchescanbeevaluatedinparallel）。