MapReduce是当前广泛采用的大数据集计算模型和框架，主要应用于海量数据的并行计算，其计算框架分为四个阶段：第一个阶段是split，主要是将大文件切分成小文件。第二个阶段是map，做些基本的分析，一般一个split对应一个map。一般很少在map端做累加，如果文件较大，就要将split传给map的文件全部加载到内存。第三个阶段是shuffle，主要做排序、分区、分组，连接map与reduce。第四个阶段是reduce，做进一步分析，reduce在拿数据的时候是通过迭代器的方式拿的，避免了内存溢出的情况。MapReduce组成：

1. Mapper

MapReduce的思想：分而治之。Mapper负责“分”即把复杂的任务分解为若干个简单的任务执行，这样数据或计算规模相对于源任务大大缩小，就近计算，即会被分配到存放了所需数据的节点进行计算，并且这些小任务可以并行计算，彼此间几乎没有依赖关系。

计算框架Mapper中resourcemanager主要是对计算流程的管理，数据存放在datanode上，计算也在这上面计算。同时，namenode管理元数据信息，过程中resourcemanager会请求namenode。

2. Shuffle

Shuffle是介于mapper与reducer中间的一个步骤，hadoop一般都是移动计算而不移动数据，但是在shuffle阶段有数据的移动。首先map以K-Value键值对的形式输出，输出后写到内存缓冲区，每一个map\_task都有一个内存缓冲区（默认100MB）存储着map的输出结果。当写入内存缓冲区中的数据达到了一定的阈值时，将缓冲区的数据以一个临时文件的方式存放到磁盘（split）。溢写是由单独线程来完成，不影响往缓冲区写map结果的线程（split.percent）默认是0.8，将溢写的过程中的一个个磁盘小文件进行分区，分区的目的是为了标记这些数据都是由后面的哪个reduce来处理。分区的默认规则是key的hash值%reduce的个数。当溢写线程启动后，需要对这80MB空间内的KEY做排序（sort）。将磁盘小文件合并成一个大文件（combiner），然后reduce主动去map端把属于自己的数据拉取过来，到了reduce端要进行二次排序（分组）。同时reduce端的数据也是加载到内存的，内存满了同样会触发溢写。过多的小文件同样会合并成大文件，最后是reduce的输出。

3. Reducer

Reducer主要是对map阶段的进行汇总，Reduce的数目由mapred-site.xml配置文件里的项目mapred.reduce.tasks决定，缺省值为1，用户可以覆盖。其中resourcemanager用于调度，reduce从map端拿数据，并进行汇总，将结果输出到hdfs中。

GFS系统由三部分组成：GFS master、GFS Client、GFS chunkserver。其中，GFS master任意时刻只有一个，而chunkserver和gfs client可能有多个。

　　一份文件被分为多个固定大小的chunk（默认64M），每个chunk有全局唯一的文件句柄——一个64位的chunk ID，每一份chunk会被复制到多个chunkserver（默认值是3)，以此保证可用性与可靠性。chunkserver将chunk当做普通的Linux文件存储在本地磁盘上。

　　GFS master是系统的元数据服务器，维护的元数据包括：命令空间（GFS按层级目录管理文件）、文件到chunk的映射，chunk的位置。其中，前两者是会持久化的，而chunk的位置信息来自于Chunkserver的汇报。

　　GFS master还负责分布式系统的集中调度：chunk lease管理，垃圾回收，chunk迁移等重要的系统控制。master与chunkserver保持常规的心跳，以确定chunkserver的状态。

GFS client是给应用使用的API，这些API接口与POSIX API类似。GFS Client会缓存从GFS master读取的chunk信息（即元数据），尽量减少与GFS master的交互。

在GFS中，master是单点，任意时刻，只有一个master处于active状态。单点简化了设计，集中式调度方便很多，也不用考虑糟心的“脑裂”问题。但是单点对系统的吞吐能力、可用性提出了挑战。那么如何避免单点成为瓶颈？两个可行的办法：减少交互，快速的failover。

　　master需要在内存中维护元数据，同时与GFS client，chunkserver交互。至于内存，问题并不大，因为GFS系统通常处理的是大文件（GB为单位）、大分块（默认64M）。每个64M的chunk，对应的元数据信息不超过64byte。而对于文件，使用了文件命令空间，使用前缀压缩的话，单个文件的元数据信息也少于64byte。

GFS client尽量较少与GFS master的交互：缓存与批量读取（预读取）。首先，允许Chunk的size比较大，这就减少了客户端想master请求数据的概率。另外，client会将chunk信息缓存在本地一段时间，直到缓存过期或者应用重新打开文件，而且，GFS为chunk分配有递增的版本号（version），client访问chunk的时候会携带自己缓存的version，解决了缓存不一致的问题。

FastSGG能够高效、可扩展地生成具有真实社会图特征的trillionscale图。首先判断是否存在满足给定模式的图是NP-complete的。因此，很难在多项式时间内生成严格满足模式的图。对于节点属性生成，可以将拓扑结构和属性值关联起来，即连接的节点往往具有相似的属性。用户可以指定属性值分布。此外，同一社区中的节点具有相似的属性值。以便FastSGG可以使用用户定义的配置生成图。

论文中提出了度分布生成(D2G)模型，用于加速图的生成过程。D2G模型采用启发式方法为源顶点生成出度和目标顶点。该模型生成外倾角或为源顶点确定目标顶点的时间复杂度为O(1)，因此可以显著提高社交图生成的速度。

社交图生成方法有两种社交图生成方法。LFR是一个广泛使用的基准，以生成具有社区结构的社会图表。程度和社区规模均符合幂律分布。LFR基于以下规则构建社区:相同社区中的顶点共享更多链接，而不同社区中的顶点共享更少的链接。然而，由于构建社区时的高计算开销，LFR不适合生成大规模图。acMark使用潜在因素生成具有社区结构的属性图。由于acMark中涉及的矩阵乘法的高计算成本，它不适合生成万亿规模的图。S3G2是一个可扩展的结构相关社交图生成器。它创建了一个包含非均匀价值分布和结构相关性的综合社会图。但它并没有考虑在由此产生的社会图表中生成社区结构。LDBC SNB从S3G2发电机演变而来。它是社交网络基准中的一种代表性方法，可以通过模拟社交网络中的用户活动来生成具有复杂依赖关系的社交图。预先定义了用户活动和属性信息。然而，所有这些社交图生成器不支持以流方式本地生成数据。