哈尔滨工业大学计算机科学与技术学院

《算法设计与分析》

课程报告

学号	L170300901
姓名	卢兑玑
报告日期	2021 年 1 月 5 日

1 论文题目

Wenfei Fan, Ping Lu, Xiaojian Luo, Jingbo Xu, Qiang Yin, Wenyuan Yu, Ruiqi Xu:

Adaptive Asynchronous Parallelization of Graph Algorithms. SIGMOD Conference 2018: 1141-1156

链接：https://dl.acm.org/doi/10.1145/3183713.3196918

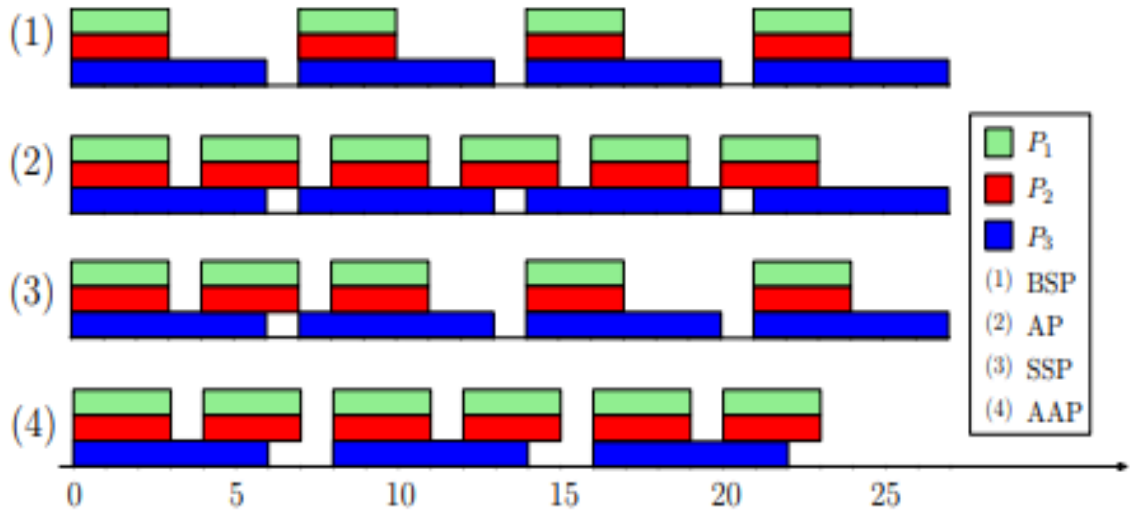
2 论文阅读报告

2.1 摘要

本文提出了一种自适应异步并行方法（AAP）用于图形计算的模型。AAP 相比于传统的方法通过减少了通过动态调整减少执行程序一次执行完到下一次开始执行的时间和减少计算过时的程序之间交互的信息来实现进程的相对进步。AAP 通过自适应切换来优化并行处理。作者采用了 GRAPE 的编程模型。此外， AAP 可以最佳地模拟 MapReduce，PRAM，BSP，AP 和 SSP。实际上 AAP 模型也具有非常好的表现结果。

2.2 问题定义

这篇论文的名字叫做自适应异步并行的图算法。在异步并行处理多个进程的过程中总是会出现有些进程落后的情况，或者是有些进程的执行需要用到以前的进程的执行结果。然而由于以前的进程执行时间比较长，这就导致了进程需要等待很长时间才能够执行。另外，每一个进程执行完毕之后将结果所得的数据传递给下一个进程也需要时间。这就是本篇论文中所关注的，如何减少这两个过程的时间开销。在这篇论文中，形象的将这个过程比喻成一个工厂里多个工人执行多个任务，每一个工人可执行的任务也是有限的，在这样的条件下，每个工人不停的执行某个任务，如何使得获得更高的生产效率。



BSP、AP、SSP、AAP 四种模型的对比

如何合理的分配，使得总消耗的时间最少。解决这样的问题以前已经有了 Bulk Synchronous Parallel (BSP) model 模型、Asynchronous Parallel (AP)模型、Stale Synchronous Parallel model (SSP)模型等来计算这样的问题。Bulk Synchronous Parallel (BSP) model 模型是一个全局同步的模型，在这个模型下，始终都是每一个工人完成都完成一个任务之后在执行下一个循环，中间经过一个间隔时间之后再执行下一个循环，这种算法在循环之间的交替过程只需要等待，因此 parallel algorithms 非常的简单，比较适合用于每个工人的工作时间的平均值和每个工人的工作时间之比都接近 1 的条件下比较适合用这个模型。为了解决 BSP 模型的问题，Asynchronous Parallel (AP)模型出现了，这是一个异步模型，在这个模型中只要每一个工人完成一份工作，就立刻执行下一个工作，然而 P3 做的第 n 次工作时候，如果 n 足够大就意味着有很多的 P1、P2 已完成的工作的信息，对于全局，信息的 update 也就变的越来越困难，在下一个 P3 的工作开始的时候计算很多 P1、P2 冗余的信息会消耗很多的代价，可见这个模型也不是一个非常好的方法。Stale Synchronous Parallel model (SSP)模型是基于 BSP 模型和 AP 模型的一个改进，这个模型允许使用 AP 模型所使用的每一个工人完成一份工作，就立刻执行下一个工作的策略，但是只能每 c step 使用一次，就是说在同步的基础上划分为许多 c 次组成的小时间段，在每个小时间段内使用 AP 的方法，这样即减少了等待滞后工人的时间，有减少了信息更新的代价，但相比于 BSP 模型信息更新的代价还是提高了。如何让两个代价都减少，就是本论文的核心 AAP 模型

2.3 算法或证明过程

下面详细的介绍 AAP 模型

1. 编程模型的理论依据

AAP 模型是在图的一个拓扑上对拓扑中的每一个元素进行操作。假设 G 是一个图，F 是图的一个拓扑，Fi 是拓扑中的一个元素。定义每一个元素入度大于 1 的顶点为边界。这个模型需要三个函数。

对于一个图，有边和节点以及权重这些参数值。这个模型算法的整体思想是通过深度优先搜索算法计算出每个顶点到其它顶点的最短权重值，将最短权重值作为更新的内容对一个个子图进行更新。

这个模型是根据参考文献 24 中提出的 GRAPE 模型来进行实现的。GRAPE 模型是一个对图的并行计算的一个处理模型，其中包括 PEval、IncEval 以及 assemble 操作。其中基本思想是首先把一个完整的图划分成子图，然后对每个字图进行处理，处理之后会有子图之间信息的交互。但是每个子图的一次处理的时间有所不同，所以处理一个图的时候就会出现上一节中所叙述的问题。所以本篇论文提出的自适应异步并行处理模型就是用来解决实现信息交互和整体效率的相结合的最优点。

Input: A fragment $F_i(V_i, E_i, L_i)$.
Output: A set $Q(F_i)$ consists of current v.cid for $v \in V_i$.
Message preamble: /*candidate set C_i is $F_i.O$*/
For each node $v \in V_i$, a variable v.cid;
1. $\mathbb{C}$:= DFS(F_i); /* find local connective components by DFS */
2. **for each** $C \in \mathbb{C}$ **do**
3. create a new "root" node v_c;
4. v_c.cid := min$\{v.\text{id} \mid v \in C\}$;
5. **for each** $v \in C$ **do**
6. link v to v_c; v.root := v_c; v.cid := v_c.cid;
7. $Q(F_i) := \{v.\text{cid} \mid v \in V_i\}$;
Message segment: $M_{(i,j)} := \{v.\text{cid} \mid v \in F_i.O \cap F_j.I, i \neq j\}$;
$f_{\text{aggr}}(v) := \min(v.\text{cid})$;

PEval：输入一个拓扑中的元素，计算这个元素的 query Q(Fi)，其中输出的结果中包含着 Fi 中每一个节点的 cid。Q(G)是每一个拓扑中元素的 query 的总和。对拓扑中每一个元素使用深度优先搜索算法计算与局部相连接的成分并为每一个节点创造出一个 id，也就是每一个顶点的 cid 的初始值。对于每一个 Fi 中每一个有出度的顶点（边界节点）的 cid 构成的集合记为 Ci，作为将来更新 query 的参数。如果计算出某一个顶点有多个 cid，就选取最小的 cid，记为 faggr(v)。PEval 创建了一个根节点 Vc 用来连接每一次 DFS 搜索到的局部联通集合中的每一个节点，根节点存储整个图中最小的 id。另外定义 M(i,j)为起点在 Fi 中出度大于 0 的顶点和在 Fj 中的入度大于 0 的顶点的交集的顶点的 cid。

Input: A fragment $F_i(V_i, E_i, L_i)$, partial result $Q(F_i)$, and message M_i.
Output: New output $Q(F_i \oplus M_i)$
1. $\Delta := \emptyset$;
2. **for each** v^{in}.cid $\in M_i$ **do** /* use min as f_{aggr} */
3. v.cid := min$\{v.\text{cid}, v^{\text{in}}.\text{cid}\}$;
4. $v_c := v$.root;
5. **if** v.cid < v_c.cid **then**
6. v_c.cid := v.cid; $\Delta := \Delta \cup \{v_c\}$;
7. **for each** $v_c \in \Delta$ **do** /* propagate the change */
8. **for each** $v \in F_i.O$ that linked to v_c **do**
9. v.cid := v_c.cid;
10. $Q(F_i) := \{v.\text{cid} \mid v \in V_i\}$;
Message segment: $M_{(i,j)} := \{v.\text{cid} \mid v \in F_i.O \cap F_j.I, v.\text{cid decreased}\}$;

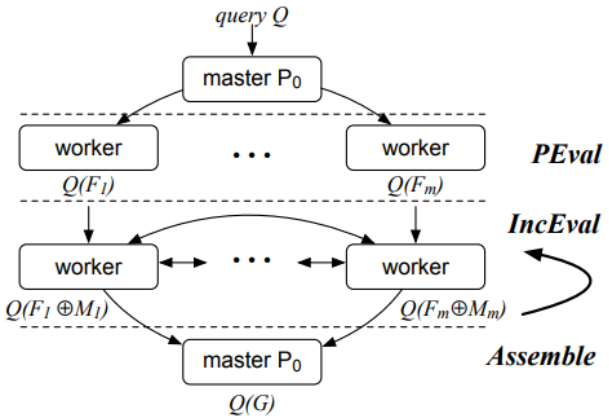
IncEval：用于对 PEval 计算出来的 query 进行更新，输入一个图、PEval 计算出的 Q(Fi)和 Mi（边界节点的 cid），输出新的 query。首先定义一个空集▲，对于每一个边界中的顶点，更新 cid（使用小的 cid），如果这个节点的 cid 小于根节点的 cid，就将根节点加入到集合中。再对每一个集合▲的每一个顶点，更新它的子节点的 cid。

Assemble：这个函数首先更新每一个与根节点连接的节点的 cid，然后将所有的有着相同的 cid 的节点放到一个集合中，并返回。

2. AAP 模型

在 AAP 模型，一个图的每一个拓扑中的元素代表一个进程，在整个体系中只有 P0 负责 Assemble 函数的操作，包括决策是否终止。上一节中 PEval 函数中的 M(i, j)就是每两个个进程之间所传递的信息，在解决这个问题的模型中还设置了如下几个概念。对每一个边界的顶点设置一个下标，用来存放这个顶点在拓扑集合中的索引，对每一个进程都设置一个缓冲区用来存储与自己有关的 M(i, j)，这样的数据结构可以保证进程与进程之间可以实时交流，信息的传递不需要排队。

另外为每一个进程都设置了一个 DSi（延迟时间），这段时间用来处理计算 cid 的更新，DSi 的大小依据两个参数，这样可以保证不会造成某一个进程不停的快速工作，导致 n 次循环之后信息的 update 开销过大的问题，一个是每一个进程缓冲区的数量，另一个是轮数的最大值和最小值，也就是在整个工作的一个大循环中允许这个进程进行的循环次数的最大值和最小值。



AAP 工作模型

如上图所示，AAP 模型工作方式如上图所示，首先是局部计算，使用 PEval 函数计算每一个工人的 query，计算出 M(i, j)，然后是更新 query 的计算，使用 IncEval 函数再对计算出的 query 进行更新，具体执行过程中如果工人的缓冲区不为空并且延迟时间已到的情况下 IncVal 函数将触发工人进行工作，首先进行边界的计算，再根据边界的顶点更新 query，最后计算新的 M(i, j)并将信息传给工人 Pj。最后是终止，每当一个工人的缓冲区变空的时候，即不再接收其他工人传递的信息，这个工人传递给 P0 一个不活跃信息，当所有的工人都传递过不活跃信息之后，如果有的工人还在工作（有可能这个工人不活跃期间接收到其他工人传递的信息后又再次活跃了起来），则程序继续执行，如果所有的工人都在不活跃状态，则程序结束，Assemble 函数在 P0 的控制下决定是否结束计算。

3. AAP 模型如何减少 cost

AAP 模型主要从三个方面减少开销。首先是减少了在信息传递方面冗余的计算，信息可以实时在工人之间传递，而且不会造成大量的信息积累，而且兼顾了工作慢的工人，通过 Assemble 函数整体控制所有的工人的工作情况以工作的进程为依据调度工人是否进行工作。IncEval 函数在局部上做计算，输入 Fi ， Q, Q(Fi) 和 Mi，计算ΔOi 使得 Q(Fi ⊕ Mi) = Q(Fi) ⊕ ΔOi，这个过程是一个 |Mi |+|ΔOi |数量级的开销，这样的计算方式可以有效的减少重复计算，因为

如果采用其他模型，就必须对所有的工人同时进行计算，也就一定会出现对某个工人重复计算了很多次的情况。PEval 和 IncVal 两个函数都是在图上执行的算法，而且都是顺序执行的算法，因此已经继承了顺序算法所有的优化策略。

4. 算法的正确性

要证明整个算法的正确性，只需要证明 PEval、IncEval 和 Assemble 三个函数的正确性，首先有三个前提条件，整个算法只是在图的一部分操作，IncEval 函数是收敛的，IncEval 函数是单调的。算法终止的条件是所有的 Query 值都不再变化，因此算法的覆盖范围可以达到所有的工人，由于 IncEval 函数的收敛性可以判断，由于 IncEval 函数对 cid 的变化只有减法操作，因此是单调递减的，由于所有的点都限定在了输入的图中，因此 IncEval 函数一定收敛，最终 Query 值一定会趋向一个定值。Assemble 函数仅仅进行了搜寻操作，因此正确性毋庸置疑。

2.4 实验结论

作者提出了 AAP 模型来弥补传统的 BSP 和 AP 模型的局限性。并且通过实验已经证明，作为异步模型，AAP 不会使编程变得更加困难，并且保留了一致性和收敛性。我们也通过实验证明了 AAP 模型的强大效率和灵活性。 我们的实验结果证明了 AAP 有望用于大规模图形计算。

3 领域综述

这篇论文所研究的问题是图的并行计算问题。目前已经出现了几种用于图形计算的并行系统，例如 Pregel，GraphLab，Giraph ++和 Blogel。 但是，这些系统要求用户将图形算法重铸到其模型中。尽管对图形进行了数十年的研究，并且已经有了许多顺序算法，但是举个例子，比如要使用 Pregel 系统，必须将现有算法重塑为以顶点为中心的模型。与其他系统编程时类似。对于不太熟悉并行模型的人来说，重铸并不容易。这使这些系统仅对有经验的使用者比较有效。

为了回答这个问题，科学家们开发了 GRAPE（并行 GRAPh 系统），用于图形计算，例如遍历，模式匹配，连接性和协作过滤。在以下方面，它与现有的图形处理系统不同。

（1）易于编程。 GRAPE 支持简单的编程模型。对于图查询的一个类，使用者只需要为 Q 提供三个顺序（增量）算法，并进行少量添加即可。无需修改现有算法的逻辑，从而大大减少了思考如何实现并行的工作量。

（2）半自动并行化。 GRAPE 基于部分评估和增量计算的组合使顺序算法并行化。如果提供的三个顺序算法正确，则可以保证在单调条件下以正确的答案终止。

（3）图级优化。GRAPE 继承了可用于顺序算法和图形的所有优化策略，例如 indexing, compression and partitioning。

（4）Scale-up。与最先进的系统（例如：以顶点为中心的 Giraph 和 GraphLab 和以块为中心的 Blogel）相比，编程的简便性并不意味着性能下降。GRAPE 在响应时间和通讯成本方面均优于 Giraph，GraphLab 和 Blogel。

4 实验和分析

4.1 论文中所用到的定理

证明本论文中的几个关键定理所用的前提条件：
(T1) 更新参数的值来自有限域 。
(T2) IncEval 是收敛的。
(T3) IncEval 是单调的。
Under AAP, a PIE program ρ guarantees to terminate with any partition strategy P if ρ satisfies conditions T1 and T2.
首先对于每个片段 Fi（i∈[1，n]），ρ在第（Ni + 1）轮终止，那么在每一轮 ri≤Ni 中，IncEval 至少会改变一个更新参数。接下来证明在 AAP 下ρ一定会终止。使用反证法假设存在一个 Q∈Q 和一个图形 G，使得ρ不终止。用（a）Nx 表示由变量 x 的赋值组成的有限集的大小，其中 x 是 G 的更新参数；（b）N 为 Nx 中元素的求和，即分配给更新参数的不同值的总和。由于ρ不终止，因此在 AAP 轨迹中存在一个运行至少 N +1 轮的工作程序 Pi。通过以上属性，在每一轮 IncEval 中，至少更新一个状态变量。因此，存在一个变量 x，该变量被更新 Nx + 1 次。此外，由于 IncEval 正在收缩，因此 x 的分配值遵循部分顺序。因此，x 必须被分配 Nx + 1 个不同的值，这与 x 仅有 Nx 个不同的值的假设相矛盾。所以假设不成立，这个过程一定会终止。
Under conditions T1, T2 and T3, AAP correctly parallelizes a PIE program ρ for a query class Q if ρ is correct for Q, with any partition strategy P.
从定理 1 可以得知，在 AAP 模型下，任何的处理程序都会终止。BSP 模型作为 AAP 模型的一个特例。假设所有的程序都在 r * 轮处理之后终止。对于 r 轮之后每一个子图的权重分析结果，我们只需要知道在 AAP 模型下到达这个点 fixpoint 的对每一个字图的分析结果。
只需要证明如下：
（1）R^{ri} i ≤ $R\tilde{}^{r}$ i 在 BSP 模型中所有的权重的中间运算结果都不比 fixpoint 点的结果小
（2）$R\tilde{}^{r}$ i ≤ R^{r} i 在 AAP 模型中所有的权重的中间运算结果都不比 fixpoint 点的结果小
到了 fixpoint 之后，所有的权重信息都不会再次更新。当所有的进程终止的时候，更新的值将会变成定值。通过反证法也可以证明这两点成立。
MapReduce and PRAM can be optimally simulated by (a) AAP and (b) GRAPE with designated messages only.
由于 MapReduce 可以模拟 PRAM，而 AAP 可以模拟 GRAPE，因此足以证明 GRAPE 可以使用指定的消息最佳地模拟MapReduce。具有 n 个处理过程的所有 MapReduce 程序都可以通过具有 n 个处理过程的 GRAPE 进行最佳模拟。 MapReduce 算法 A 定义如下。它的输入是“键，值”对的多组 IO，操作是子例程的序列（B1，..，Bk），其中 Br（r∈[1，k]）由映射器 μr 和减速器ρr。给定 IO，A 迭代运行 Br（r∈[1，k]）。用 Ir 表示子例程 Br（r∈[1，k]）的输出。映射器μr 一对一地处理 Ir-1 中的每个“键，值”对，并生成“键，值”对的多组 I′r 作为输出。通过键值对 I′r 中的组对进行分组，即，当且仅当两个对具有相同的密钥值时，

它们才在同一组中。通过不同的键对 I'r 进行分组。让 Gkl，。 。 。 ，而 Gkj 是获得的组。归约器ρr 逐个处理组 Gkl（l∈[1，j]），并生成⟨key，value⟩对的多组 Ir 作为输出。如果 r <k，则 A 以与步骤（i）-（iii）相同的方式在 Ii 上运行下一个子例程 Br + 1；否则，A 输出 Ik 并终止。

指定 PIE 程序 B 如下。

PEval 模拟子程序 B1 的映射器 μ1。

每个进程都在其本地数据上运行 B1 的映射器 μ1。

计算 μ1 的输出（I1）'，并将其存储在更新参数中，以供以后的 superstep 使用。

对于（I1）′ 中的每对“键，值”，其在更新参数中包括元组“ 1，键，值”。

如果减速器ρ1 的程序 Pi 要处理“ key，value”对，PEval 将“ 1，key，value”加到节点 wi 的更新参数上。聚合函数首先对所有 wi（i∈[1，n]）的更新参数进行并集，然后按键对元组进行分组。

IncEval 首先从接收到的“ r，键，值”元组中提取索引 r，然后使用 r 选择正确的子例程。然后，IncEval 执行以下操作：

从接收到的“键，值，值”元组的消息中提取“键，值”对的多集（Ir）'

在（Ii）'上运行被视为 IncEval 的分支程序的减速器ρr；用 Ir 表示ρr 的输出;

如果 r = k，则 IncEval 将更新的参数设置为空，从而终止 B；否则，IncEval 在 Ir 上运行映射器 μr + 1，构造元组⟨r+1，键，值⟩，每个⟨键、值⟩，μr+ 1 的输出中的值对，并以与 PEval 相同的方式分配更新参数。

Assemble 过程将所有进程的部分结果合并。可以很容易地验证 PIE 程序 B 是否正确模拟了 MapReduce 程序 A。此外，如果 A 在 T 时间运行造成信息传递成本 C，则 B 将花费 O（T）时间并发送 O（C）数量的数据。形式上，这可以通过对 k 中的 A 中子例程（B1，...，Bk）的数量进行归纳来验证。

4.2 实验验证

1. 实验设置

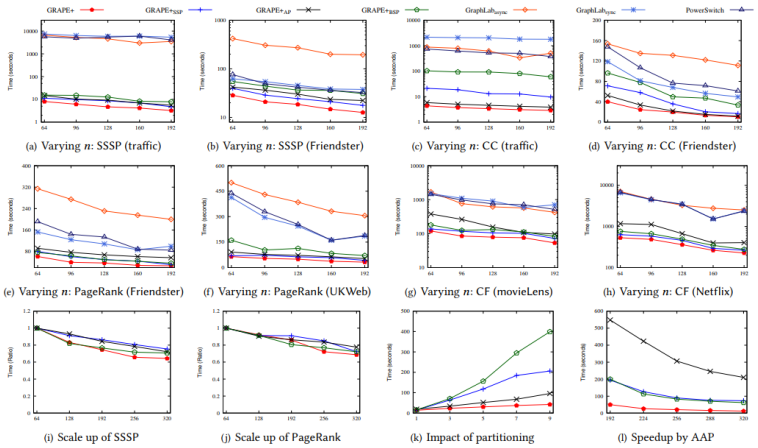
实验环境：

作者将系统部署在 HPC 集群上。总共进行了 5 个实验，对于每一个实验，都进行了 5 次测试，每次实验都使用了 20 个处理器，每个服务器有 16 个 2.40GHz 的线程，并且佑 128GB 的内存。在每个线程上都部署了一个工人。使用了 GRAPE 进行编程。同时对 GRAPE+BSP、GRAPE+AP、GRAPE+SSP 和 GRAPE+AAP 模型进行测试。实验同时分析了算法的效率，工人之间交流所消耗的代价和 AAP 模型的优势。设定一个图生成器生成图作为输入数据。

数据：

一共输入了 5 个图进行测试，分别是 6500 万用户和 18 亿条关系组成的社交网络图，具有 2300 万顶点和 5800 万条边的美国公路网络，具有 1.33 亿顶点和 50 亿条边的英国互联网图，在 138000 观众和 27000 部电影之间的有 2000 万条边的电影收视率图，以及在 480000 观众和 17770 部电影之间的有 1 亿万条边的电影收视率图。

2. 实验结果



不同模型的效率对比分析

在效率方面的实验结果显示 AAP 模型可以大幅度的提高效率。GRAPE +在所有情况下始终优于其它的模型，在社交关系图（第一个图）中生成了 192 名工人，执行速度分别比同步的 GraphLabsync，异步 GraphLabasync 和混合 PowerSwitch 快 1673 倍，1085 倍和 1270 倍

GRAPE +的性能提升来自以下方面：

（i）通过动态调整相对来有效利用资源，在 AAP 模型下工人的进步;

（ii）通过使用增量 IncEval 减少冗余计算和通信

（iii）继承自顺序算法策略的优化。

RAPE + BSP，GRAPE + AP 和 GRAPE + SSP 仍然可以通过（ii）和（iii）来实现性能的提升。实验结果表示，在社交关系图中，GraphLabsync 总共进行了 34 轮迭代，然而通过使用 IncEval 函数，GRAPE + BSP 和 GRAPE + SSP 只进行了获得 21 轮和 30 轮迭代，从而减少了同步每个工人的信息的代价。

在关系图网络中 GRAPE +平均分别比 GRAPE + BSP，GRAPE + AP 和 GRAPE + SSP 快 2.42、1.71 和 1.47 倍，最高时可分别达到 2.69、1.97 和 1.86 倍。自 GRAPE +，GRAPE + BSP，GRAPE + AP 和 GRAPE + SSP 在不同的模式下是相同的系统，差距反映了不同模型的有效性。空闲等待时间 AAP 分别为 BSP 和 SSP 的 32.3%和 55.6%。AAP 的计算时间与 AP 和 SSP 相比时间只有 37.2%和 47.1%，这些都可以证明 AAP 的效率更高。

在减少信息同步的时间开销方面，GRAPE +信息的输出量相比于 GraphLabsync，GraphLabasync 和 PowerSwitch 平均分别增加了 22.4%，8.0%和 68.3%，这是因为 GRAPE +减少了冗余的计算量以及只通过 IncEval 函数来更新参数值。GRAPE +相比于 GRAPE + BSP，GRAPE + AP 和 GRAPE + SSP 的信息交流成本分别为 1.22 倍，40%和 1.02 倍。