Adaptive Asynchronous Parallelization of Graph Algorithms

Wenfei Fan[1,2,3], Ping Lu[2], Xiaojian Luo[3], Jingbo Xu[2,3], Qiang Yin[2],Wenyuan Yu[2,3], Ruiqi Xu[1]

[1]University of Edinburgh [2]BDBC, Beihang University [3]7 Bridges Ltd.

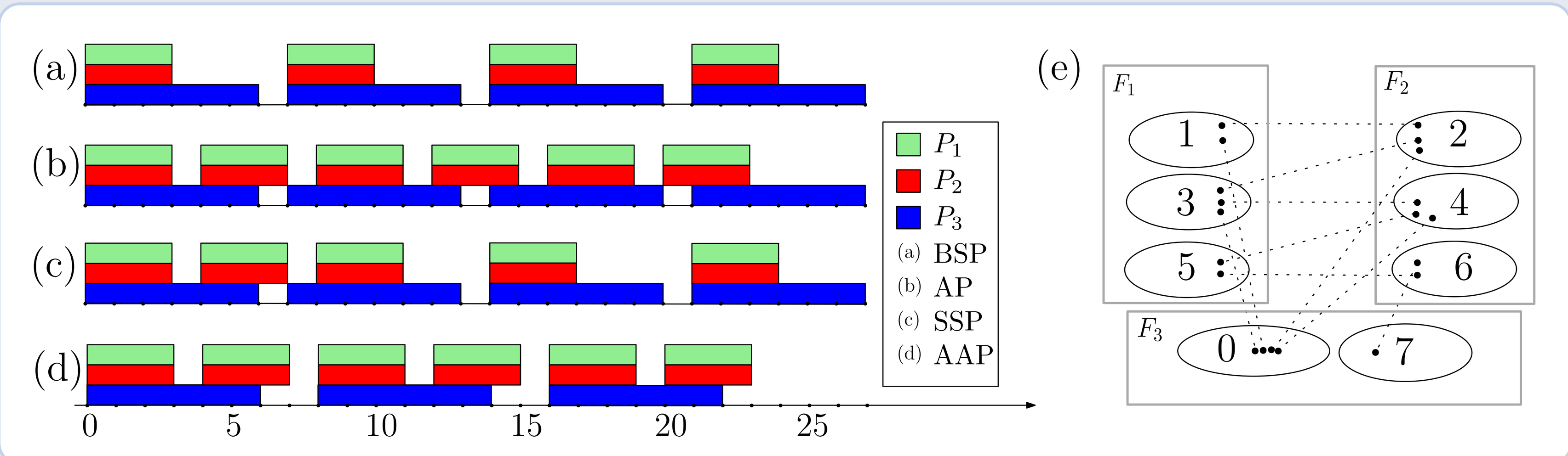
7Bridges

INTRODUCTION

We proposes an Adaptive Asynchronous Parallel (**AAP**) model for graph computations. As opposed to Bulk Synchronous Parallel (**BSP**) and Asynchronous Parallel (**AP**) models, AAP reduces both stragglers and stale computations by dynamically adjusting relative progress of workers.

- **BSP**, **AP** and Stale Synchronous Parallel model (**SSP**) are special cases of **AAP**.
- Better yet, **AAP** optimizes parallel processing by adaptively switching among these models at different stages.
- Employing the programming model of GRAPE, **AAP** parallelizes existing sequential algorithms based on fixpoint computation with partial and incremental evaluation.
- Under a monotone condition, **AAP** guarantees to converge at correct answers.
- It can optimally simulate MapReduce/PRAM/**BSP**/**AP**/**SSP**.

MOTIVATION



- **BSP**: stragglers keep other workers idle, thus wasting resources.
- **AP**: with redundant computation and communication
- **SSP**: ad-hoc staleness, incremental improvement over **BSP/AP**
- Is it possible to have a simple parallel model that inherits the benefits of BSP and AP, and reduces both stragglers and stale computations, without explicitly switching between the two?

PROGRAMMING MODEL

Data partitioned parallelism (shared-nothing architecture).

Fragmented graph G = $(G_1, \ldots, G_n)$, distributed to workers.

GRAPE API: three core functions for a graph query class Q

PEval: a (existing) sequential algorithm for Q, for partial evaluation;

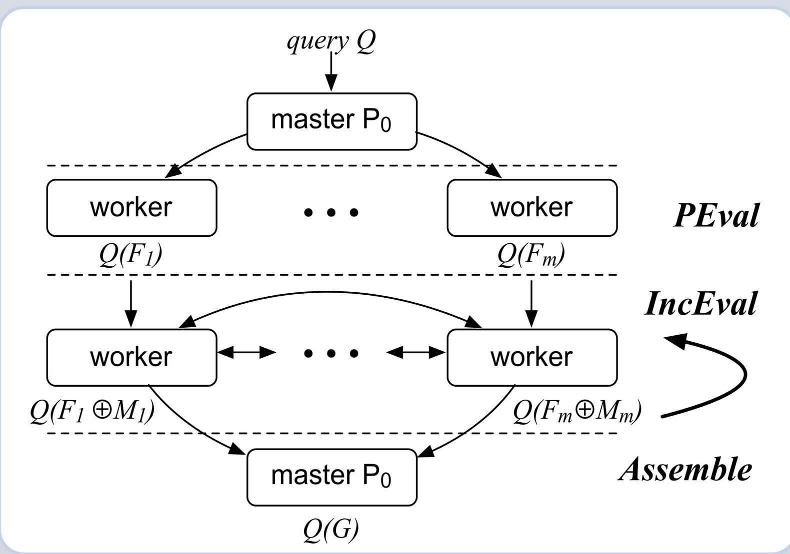
IncEval: a (existing) sequential incremental algorithm for Q;

Assemble: a sequential algorithm (taking a union of partial results).

A fixpoint computation

$\Phi\ (R_1, \ldots, R_n)$

R_i^0 = PEval(Q, G_i)
R_i^{r+1} = IncEval(Q, R_i^r, G_i, M_i)



- **Convergence guarantee:**
  - **T1**: Update parameters take values from a finite domain
  - **T2**: IncEval is contracting (the same run)
  - **T3**: IncEval is monotonic (different runs)

T1 + T2: termination

T1 + T2 + T3: the Church-Rosser property (all asynchronous runs converge at the same correct result)

- **Simulation Theorem:** MapReduce, BSP, AP, SSP and PRAM models are optimally simulated by AAP

DYNAMIC ADJUSTMENT

$$\mathrm{DS}_i = \begin{cases} +\infty & \neg S(r_i, r_{\min}, r_{\max}) \vee (\eta_i = 0) \\ T^i_{\mathrm{L}_i} - T^i_{\mathrm{idle}} & S(r_i, r_{\min}, r_{\max}) \wedge (1 \le \eta_i < \mathrm{L}_i) \\ 0 & S(r_i, r_{\min}, r_{\max}) \wedge (\eta_i \ge \mathrm{L}_i) \end{cases}$$

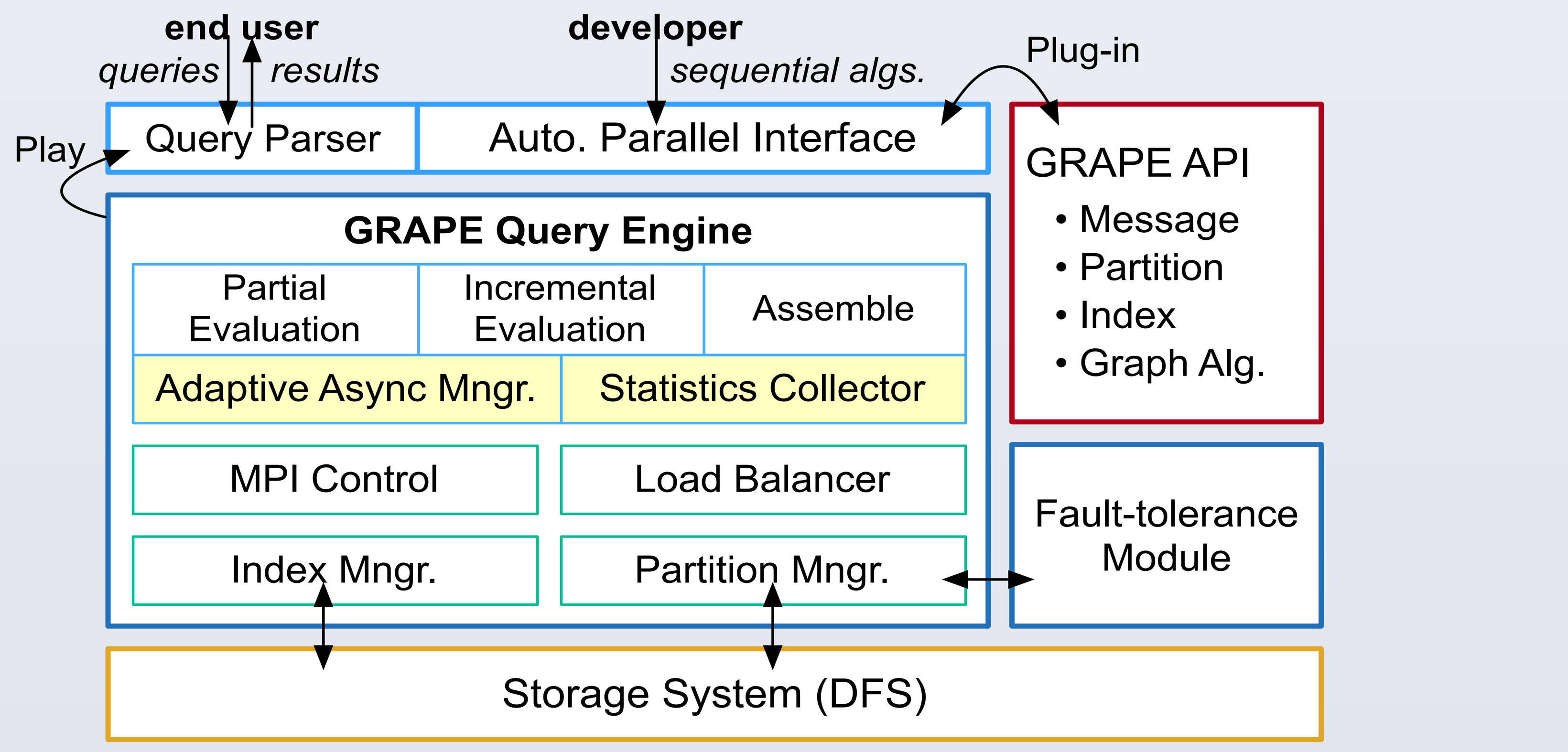
$S(r_i, r_{\min}, r_{\max})$: whether P_i should be suspended immediately.

L_i "predicts" how many messages should be accumulated,

$T^i_{\mathrm{L}_i}$ estimates how longer P_i should wait to accumulate L_i messages.

T^i_{idle} is the idle time of worker P_i after the last round.

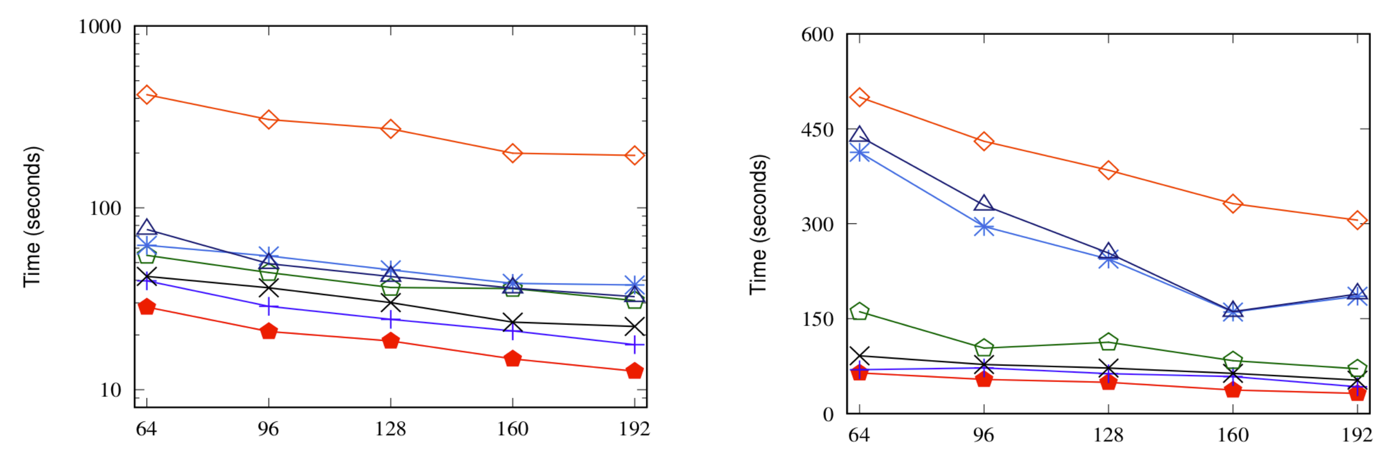
IMPLEMENTATION



PERFORMANCE

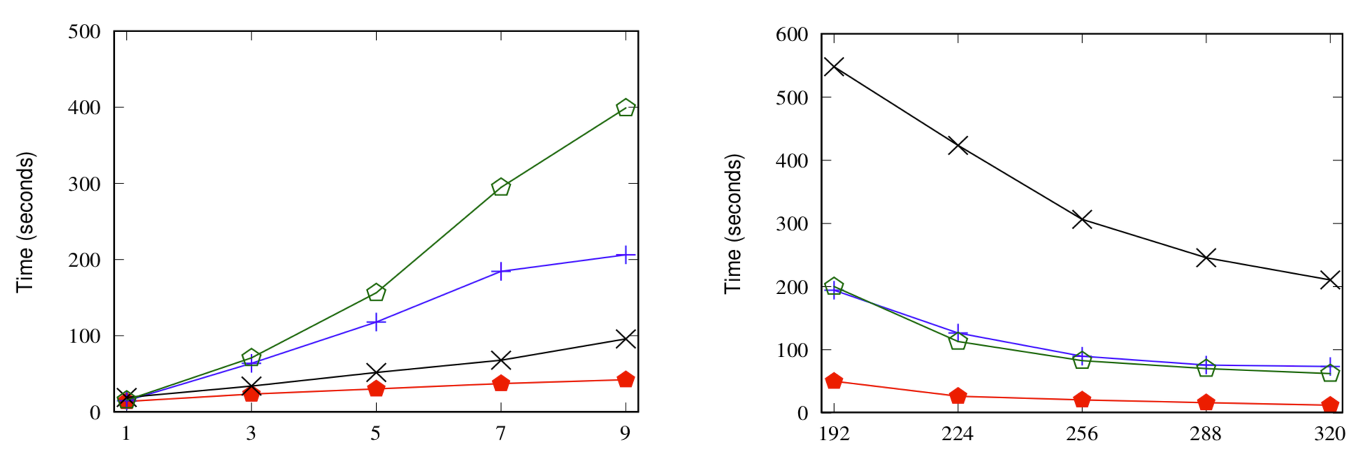
Algorithms and their applications: (1) **SSSP** (traffic analysis); (2) **Connected Component** (social analysis); (3) **PageRank**; (4) **Collaborate Filtering** (recommendation, machine learning).

Outperform BSP, AP and SSP by 4.3X, 14.7X and 4.7X on average
Outperform the state-of-the-art systems



(b) Varying n: SSSP (Friendster) (f) Varying n: PageRank (UKWeb)

Speedup by AAP in large-scale setting
up to 5.0, 16.8 and 5.9 times with 320 workers



(k) Impact of partitioning (l) Speedup by AAP

CONCLUSION

We have proposed AAP to remedy the limitations of BSP and AP by reducing both stragglers and redundant stale computations. More information about GRAPE: https://7bridges.io