Zhejiang University-University of Illinois at Urbana Champaign Institute

Unbounded Barrier-Synchronized Concurrent ASMs for Effective MapReduce Processing on Streams

Zilinghan Li[1], Shilan He[1], Yiqing Du[1], Senén González[2], Klaus-Dieter Schewe[1]

[1] Zhejiang University, UIUC Institute, Haining, China
[2] P&T Connected, Linz, Austria
{zilinghan.18|shilan.18|yiqing.18|kd.schewe}@intl.zju.edu.cn, ulcango@gmail.com

ZJUI

Contents

Introduction

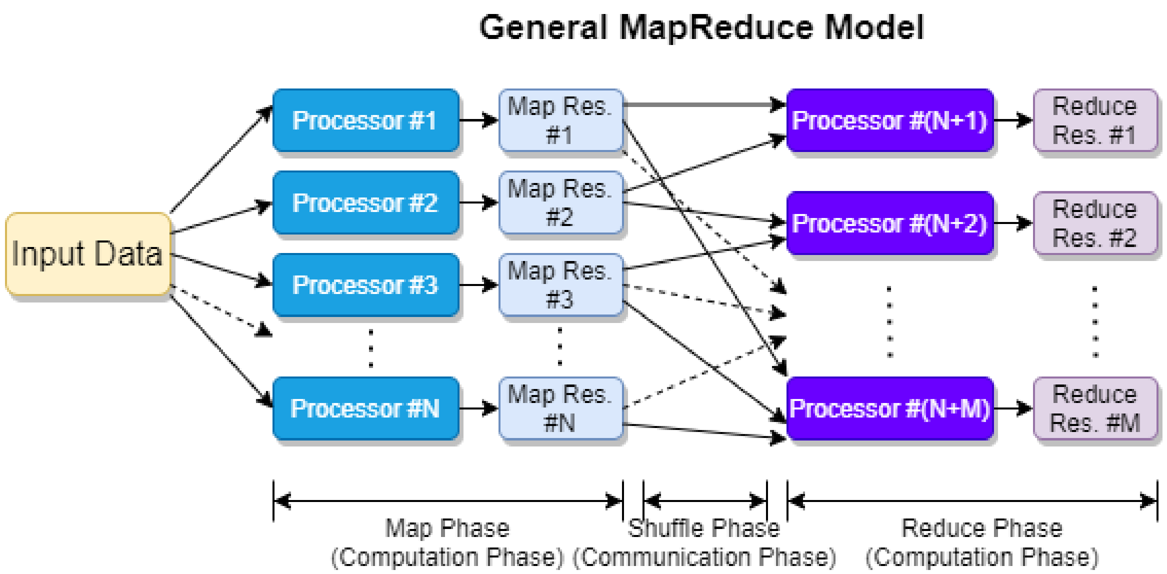
- MapReduce Algorithm [1]
  - Processing large but finite data sets in an asynchronous and distributed way.
  - Comprising *map* phase, *shuffle* phase and *reduce* phase.

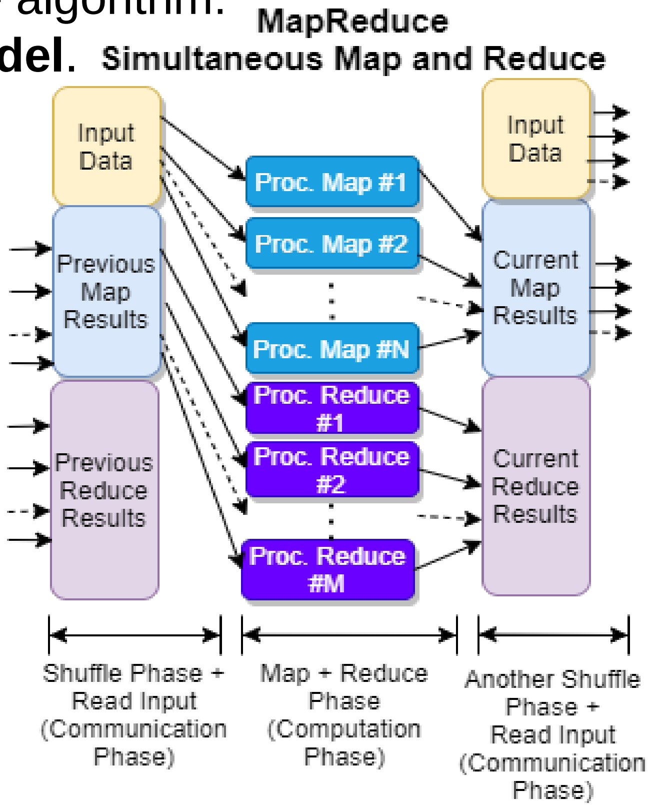
- Bulk Synchronous Parallel (BSP) Bridging Model [2]
  - Model for parallel computation.
  - Finite and fixed number of agents.
  - Comprising a sequence of *supersteps*, and each *superstep* is composed of a computation phase and communication phase.
  - The synchronization method is barrier-synchronization.

[1] Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. In: Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation – Volume 6. pp. 10–10
[2] Valiant, L.G.: A bridging model for parallel computation. Commun. ACM 33(8), 103–111 (1990). https://doi.org/10.1145/79173.79181

Introduction

➢ Connection between BSP model and MapReduce algorithm: **MapReduce can be realized based on BSP model.**





Introduction

- Stream query $Q: Stream \rightarrow Stream$ is *abstractly compuatable* if there exists a kernel function $K: finStream \rightarrow finStream$ such that Q can be obtained by concatenating ($\odot$)^1 the results of kernel function K applied to larger and larger prefixes of the input. [1]

$$Q : \mathbf{s} \mapsto \bigodot_{k=0}^{\mathrm{size}(\mathbf{s})} K(\mathbf{s}^{\leq k}),$$

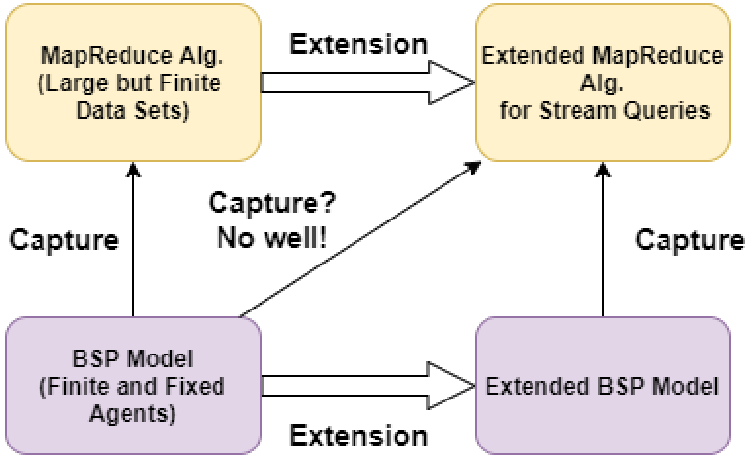
- Behavioral Theory: It consists of three parts
  - An axiomatization of a class of algorithms
  - A definition of an abstract machine model
  - A proof that the abstract machine model captures the class of algorithms stipulated by the axiomatization.

1. The concatenation ($\odot$) used here is not same as the common concatenation denoted by $\sum$. It works more like aggregation and its real functionality varies among different scenarios, but we still use the term concatenation to be consistent with the initial literature.

[1] Gurevich, Y., Leinders, D., Van den Bussche, J.: A theory of stream queries. In: Arenas, M., Schwartzbach, M.I. (eds.) DBPL 2007. LNCS, vol. 4797, pp. 153–168. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75987-4_11

Research Question

- Primary Question: Can MapReduce algorithm, based on BSP model, be used to handle stream, which is assumed to come indefinitely?
- Following Question: After solving the primary question, we find that the general BSP model is not sufficient for some stream queries, which brings about another question on how to extend the general BSP model.


MapReduce Alg. (Large but Finite Data Sets)
Extension
Extended MapReduce Alg. for Stream Queries
Capture
Capture? No well!
Capture
BSP Model (Finite and Fixed Agents)
Extension
Extended BSP Model

Methodology: Stream Query Classification

- When handling stream queries, queries may require some information on previous stream.

- According to the different requirements for previous data, we classify stream queries into three different classes, and each class has its own MapReduce model.

- Three classes of stream queries:
  - *Memoryless*
  - *Semi-Memoryless*
  - *Memorable*

Memoryless Stream Queries

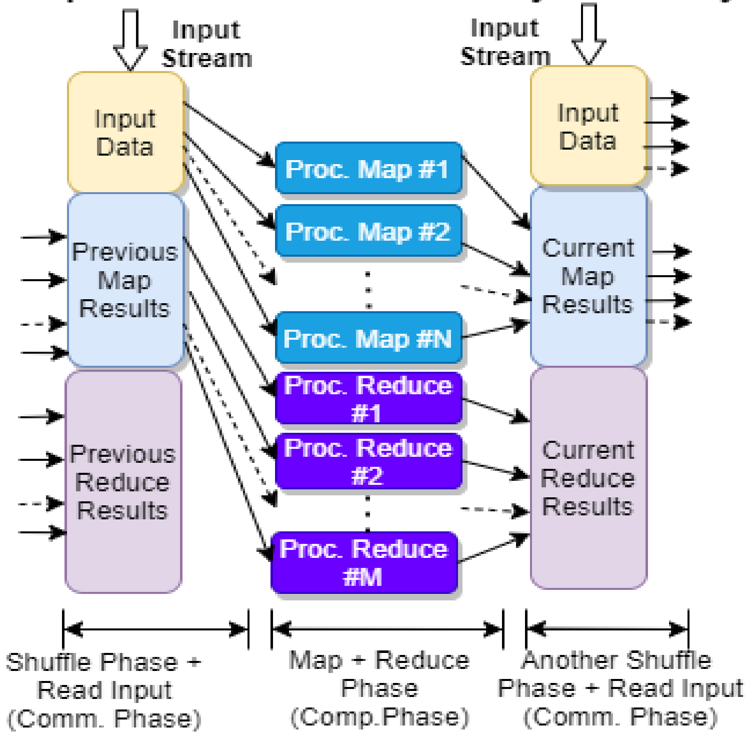
- Property: Results of new coming stream are independent of previous input streams, and the outputs of larger stream can be obtained by direct concatenation of the outputs of previous streams.
- Mathematically, $\mathcal{Q} : \mathbf{s} \mapsto \bigodot_{k=0}^{\mathrm{size}(\mathbf{s})} K(\mathbf{s}^{\leq k}) = \bigodot_{k=0}^{\mathrm{size}(\mathbf{s})} K(s_k)$
- Example: Query $\mathcal{Q}_1$ returns an output stream containing all numbers greater than a threshold value t_x. For this query, nothing about previous stream is needed when handling the current input stream.

ZJUI

Memoryless Stream Queries

- MapReduce Model: Since there are no dependency on previous input, we can simply adapt the general MapReduce model based on BSP.

MapReduce Model for Memoryless Query



$$\begin{array}{ll}
\textbf{IF} & task_j = \text{"map"} \\
\textbf{THEN} & map\_out_j := Map\_Function(\mathbf{s}_j) \\
 & bsp\_send(map\_out_j) \\
 & bsp\_sync() \\
\textbf{IF} & task_j = \text{"reduce"} \\
\textbf{THEN} & map\_out_j := bsp\_get() \\
 & reduce\_out_j := concat(reduce\_out_j, map\_out_j) \\
 & bsp\_sync()
\end{array}$$

Semi-Memoryless Stream Queries

- Property: To obtain the result of new coming stream, in addition to the outputs of previous streams, some aggregated information of previous streams, denoted as $\mathbf{I}(\boldsymbol{s}_{\boldsymbol{prev}})$, is also required.

- Mathematically, denote previous stream as $\boldsymbol{s}_{\mathbf{1}}$, current stream as $\boldsymbol{s}_{\mathbf{2}}$:

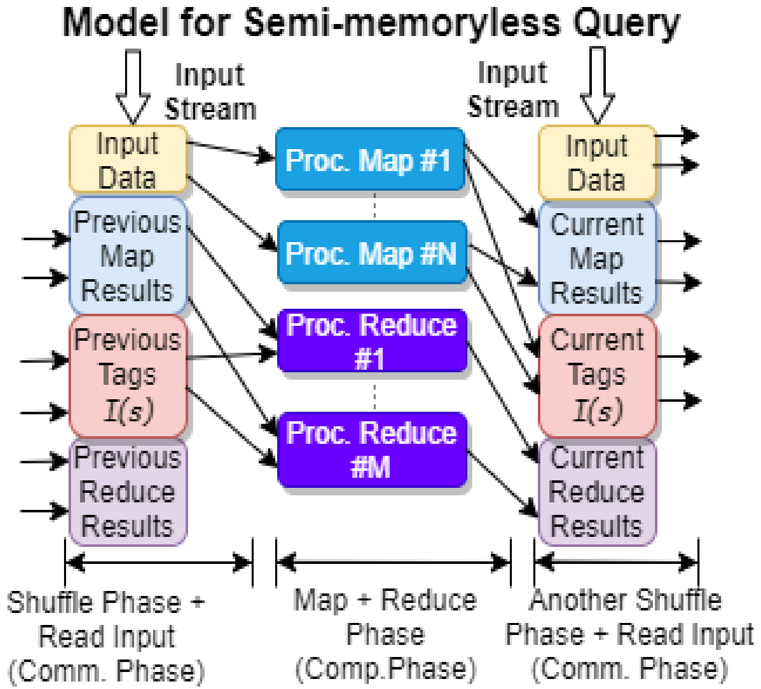
$$\mathcal{Q} : \mathbf{s} \rightarrow \mathbf{F}_{agg}(K(\mathbf{s}_1), K(\mathbf{s}_2), \mathbf{I}(\mathbf{s}_1), \mathbf{I}(\mathbf{s}_2)) \text{ with } \mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$$

- Example: Query $\mathcal{Q}_2$ returns the running average value of the numbers arrived so far. For $\mathcal{Q}_2$, we not only need to know the average of previous stream $avg(\boldsymbol{s}_1)$, we also need its length. Namely, $\mathbf{I}(\boldsymbol{s}_1) = \{len(\boldsymbol{s}_1)\}$.

ZJUI

Semi-Memoryless Stream Queries

- MapReduce Model: The processors in Map phase should not only give the result of kernel/map function $K(\boldsymbol{s}_i)$, it should also give the information set (or called tag) $\mathbf{I}(\boldsymbol{s}_i)$.



```
IF   $task_j$ = "map"
THEN   $map\_out_j := Map\_Function(\mathbf{s}_j)$
       $tag\_out_j := Tag\_Fuction(\mathbf{s}_j)$
       $bsp\_send(map\_out_j)$
       $bsp\_send\_tag(tag\_out_j)$
       $bsp\_sync()$
IF   $task_j$ = "reduce"
THEN   $map\_out_j := bsp\_get()$
       $tag\_out_j := bsp\_get\_tag()$
       $reduce\_out_j := Agg(reduce\_out_j, map\_out_j, local\_tag_j, tag\_out_j)$
       $bsp\_sync()$
```

Memorable Stream Queries

- Property: To obtain the result of new coming stream, in addition to the outputs of previous streams, a **"large"** set of information, or even the whole previous input stream $\boldsymbol{s}$ is required.
- Memorable query is just a special semi-memoryless query with the cardinality of information set $\mathbf{I}(\boldsymbol{s_i})$ in the Θ-class[1] of the cardinality of the stream $\boldsymbol{s_i}$.
- Mathematically,

$$\mathcal{Q} : \mathbf{s} \rightarrow \mathbf{F}_{agg}(K(\mathbf{s}_1), K(\mathbf{s}_2), \mathbf{I}(\mathbf{s}_1), \mathbf{I}(\mathbf{s}_2)) \text{ with } \mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2$$

$$|\mathbf{I}(\mathbf{s}_1)| \in \Theta(|\mathbf{s}_1|) \text{ and } |\mathbf{I}(\mathbf{s}_2)| \in \Theta(|\mathbf{s}_2|)$$

1. Θ-Class is the intersection of O-Class and Ω-Class which provides an asymptotically **tight** bound for functions. For example, $3x$ is in the $O(x^2)$ and $\Theta(x)$, but NOT in $\Theta(x^2)$, since x^2 is not the **tight** bound of $3x$.

Memorable Stream Queries

- Example: Query Q_3 returns the *median* of the numbers arrived so far. Given any sequence of numbers, every single number can be the candidate of the median, which means that the whole stream $\boldsymbol{s}$ is required to determine the median. Therefore, when denoting previous stream as $\boldsymbol{s_1}$ and current stream as $\boldsymbol{s_2}$, we have $\mathbf{I}(\boldsymbol{s_1}) = \boldsymbol{s_1}$, and $\mathbf{I}(\boldsymbol{s_2}) = \boldsymbol{s_2}$.
- MapReduce Model: Since Memorable query is simply a special type of semi-memoryless query, the MapReduce model is same.
- Introduced Problems: When handling memorable stream query via MapReduce, it requires a large amount of computing resources and a large storage space. In this case, the BSP model, which has finite and fixed number of agents, does not capture it well.

Methodology: Extended BSP Model (Inf-Ag-BSP)

- The extended BSP Model is named as Infinite-Agent-BSP (Inf-Ag-BSP) Model.

- Features of Inf-Ag-BSP Model:
    - Inherit most features of general BSP Model;
    - Additional feature 1: Total number of available agents can be countably infinite[1];
    - Additional feature 2: The agents can *join* or *leave* the set of active agents in the BSP computation.

1. In theory, we can assume that the number is countably infinite, provided we restrict the model such that only finitely many of them will be simultaneously active.

Inf-Ag-BSP Model

- Handling additional features:
  - (1) Modify the number of available agents to countably infinite with other corresponding modifications: It is feasible since BSP computation is restricted concurrent computation, which allows infinite number of agents in total.
  - (2.1) Introducing local variable $join_j$ to the local signature of each algorithm, and a joining set $\mathcal{J}$.
  - (2.2) Agent should only join in the communication phase, and leave in the computation phase.

ZJUI

Inf-Ag-BSP: Signature Restriction

- Signature[1] Restriction

A concurrent algorithm $\mathcal{B} = \{(a_i, \mathcal{A}_i) | i \in \mathbb{N}\}$ *satisfies the Inf-Ag-BSP signature restriction* iff the signature of $\mathcal{A}_0$ is $\Sigma_0 = \{bar_i, join_i \mid i \in \mathbb{N}\backslash\{0\}\} \cup \{barrier\}$, and the signature of any other $\mathcal{A}_i$ $(i \in \mathbb{N}\backslash\{0\})$ is $\Sigma_i = \Sigma_{i,loc} \cup \{c_{i,j}, c_{j,i} \mid j \in \mathbb{N}\backslash\{0\}, j \neq i\} \cup \{barrier\}$ such that the following conditions are satisfied:

(1) The subsignatures $\Sigma_{i,loc}$ are pairwise disjoint, and each of them contains the corresponding function symbols bar_i and $join_i$ of arity 0;
(2) For $\mathcal{A}_i$, all locations $(c_{i,j}, \boldsymbol{v})$ are write-only;
(3) For $\mathcal{A}_i$, all locations $(c_{j,i}, \boldsymbol{v})$ are read-only;
(4) The function symbol $barrier$ has arity 0 and is monitored for all agents a_i $(i \neq 0)$;
(5) For $\mathcal{A}_0$, it monitors bar_i and $join_i$.

1. A signature Σ is a finite set of function symbols, and each $f \in \Sigma$ is associated with arity $ar(f) \in \mathbb{N}$.

Inf-Ag-BSP: Communication-and-Joining Postulation

For a concurrent algorithm $\mathcal{B} = \{(a_i, \mathcal{A}_i) \mid i \in \mathbb{N}\}$ with *concurrent run* $S_0, S_1, \cdots$, update sets $\Delta_n = \bigcup_{a \in Ag_n} \Delta_a(res(S_{a(n)}, \Sigma_a))$ for $S_{n+1} = S_n + \Delta_n$, and a joining set $\mathcal{J}_n = \{i \mid \mathrm{val}_{S_{a_i(n)}}(join_i) = \textbf{true}\}$, $\mathcal{B}$ *satisfies the Inf-Ag-BSP communication-and-joining postulate* iff the following conditions are satisfied:

(1) The cardinality of joining set $\mathcal{J}_n$ is finite for all n;
(2) If $i \notin \mathcal{J}_n$, then either $a_i \notin Ag_n$ or $(join_i, \textbf{true}) \in \Delta_n$;
(3) If there exists an update $(l, v) \in \Delta_n$ with a location of form $l = (c_{i,j}, \boldsymbol{v})$, then all updates in Δ_n(except $(bar_x$,**false**$)$ and $\{(bar_y, \textbf{true}), (join_y, \textbf{true})\})$ have this form and $\mathrm{val}_{S_{a(n)}}(barrier) = \textbf{true}$ holds for all $a \in Ag_n$;
(4) If there exists no update $(l, v) \in \Delta_n$ with a location of the form $l = (c_{i,j}, \boldsymbol{v})$, then $\mathrm{val}_{S_{a(n)}}(barrier) = \textbf{false}$ holds for all $a \in Ag_n$;
(5) If non-trivial update $(join_i, \textbf{true}) \in \Delta_n$, then $(bar_i, \textbf{true}) \in \Delta_n$;
(6) If non-trivial update $(join_i, \textbf{false}) \in \Delta_n$, then $\mathrm{val}_{S_{a_i(n)}}(bar_i) = \textbf{false}$;
(7) Whenever $(bar_i \wedge \neg barrier) \vee (\neg bar_i \wedge barrier)$ holds for any agent $i \in \mathcal{J}_n$ in state S_n, then $\Delta_{a_i(n)}(res(S_n, \Sigma_i)) = \emptyset$;
(8) The location *barrier* is lazy, namely, it only changes value when all bar_i for $i \in \mathcal{J}_n$ have changed their values from **true** to **false** (or **false** to **true**).

ZJUI

Inf-Ag-BSP Abstract State Machine: Rules

- Intuitive interpretation for rules:
  - $\mathit{process\ rules}$: rules in computation phase
  - $\mathit{barrier\ rules}$: rules in communication phase
  - $\mathit{join\ rules}$: rules for inactive agents to join in the BSP computation
- Rules for Inf-Ag-BSP Abstract State Machine:
  - $\mathit{process\ rule}$: Assignment rule $f(t_1, \dots, t_n) \coloneqq t_0$ with $f \in \Sigma_{i,loc} \setminus \{bar_i, join_i\}$ is a $\mathit{process\ rule}$. Assignment rules $bar_i \coloneqq \mathbf{true}$ and $join_i \coloneqq \mathbf{false}$ are also $\mathit{process\ rules}$.
  - $\mathit{barrier\ rule}$: Rule $c_{i,j}(t_1, \dots, t_n) \coloneqq t_0$ with $j \neq i$ is a $\mathit{barrier\ rule}$. Rule $bar_i \coloneqq \mathbf{false}$ is also a $\mathit{barrier\ rule}$.
  - $\mathit{join\ rule}$: The only possible $\mathit{join\ rule}$ is the parallel composition of $join_i \coloneqq \mathbf{true}$ and $bar_i \coloneqq \mathbf{true}$.
  - The parallel composition $(r_1 | r_2 \dots | r_m)$ and conditional composition $(\mathbf{IF}\ \varphi\ \mathbf{THEN}\ r)$ of $\mathit{process\ rule}$ (or $\mathit{barrier\ rule}$ respectively) is also $\mathit{process\ rule}$ (or $\mathit{barrier\ rule}$ respectively).

Inf-Ag-BSP Abstract State Machine

- Definition: Inf-Ag-BSP Abstract State Machine
  - Concurrent ASM $\{(a_i, \mathcal{M}_i) | i \in \mathbb{N}\}$
  - Rule for $\mathcal{M}_0$: used to switch between computation and communication phase. $\mathcal{J}$ is a finite set with $\mathcal{J} = \{i \mid val(join_i) = \textbf{true}\}$.

$$
\begin{aligned}
&\texttt{SWITCH} \equiv \\
&(\text{IF } \neg barrier \wedge \bigwedge_{i \in \mathcal{J}} bar_i \text{ THEN } barrier := \textbf{true}) \mid \\
&(\text{IF } barrier \wedge \bigwedge_{i \in \mathcal{J}} \neg bar_i \text{ THEN } barrier := \textbf{false})
\end{aligned}
$$

  - Rule for $\mathcal{M}_i$ $(i \neq 0)$: Inf-Ag-BSP rule defined as below. $r_{i,proc}$ is *process rule*, $r_{i,comm}$ is *barrier rule* and $r_{i,join}$ is *join rule*.

$$
\begin{aligned}
&(\text{IF } join_i \wedge \neg bar_i \wedge \neg barrier \text{ THEN } r_{i,proc}) \mid \\
&(\text{IF } join_i \wedge bar_i \wedge barrier \text{ THEN } r_{i,comm}) \mid \\
&(\text{IF } \neg join_i \text{ THENCHOOSE } r_{i,join} \text{ OR } skip)
\end{aligned}
$$

Inf-Ag-BSP: Characterization Theorem

- Characterization theorem
  - (1) Inf-Ag-BSP-ASM satisfies the signature restriction and communication-and-joining postulate in the axiomatization
  - (2) Every Inf-Ag-BSP algorithm defined by the axiomatization can be simulated by an Inf-Ag-BSP-ASM.

MapReduce for Memoryless Query via Inf-Ag-BSP Model

- Compared with general BSP model, the main difference is allowing join and leave behavior.

General BSP Model

```
IF    $task_j$ = "map"
THEN  $map\_out_j := Map\_Function(\mathbf{s}_j)$
      $bsp\_send(map\_out_j)$
      $bsp\_sync()$
IF    $task_j$ = "reduce"
THEN  $map\_out_j := bsp\_get()$
      $reduce\_out_j := concat(reduce\_out_j, map\_out_j)$
      $bsp\_sync()$
```

Inf-Ag-BSP Model

```
IF    $join_j$ = true
THEN  $join_j := Join\_Update()$
              // Update the value of $join_i$ in current superstep
      IF    $join_i$ = true
            IF    $task_j$ = "map"
            THEN  $map\_out_j := Map\_Function(\mathbf{s}_j)$
                  $bsp\_send(map\_out_j)$
                  $bsp\_sync()$
            IF    $task_j$ = "reduce"
                  $map\_out_j := bsp\_get()$
                  $reduce\_out_j := concat(reduce\_out_j, map\_out_j)$
                  $bsp\_sync()$
```

MapReduce for Semi-Memoryless Query via Inf-Ag-BSP Model

General BSP Model

```
IF    $task_j$ = "map"
THEN  $map\_out_j := Map\_Function(\mathbf{s}_j)$
      $tag\_out_j := Tag\_Fuction(\mathbf{s}_j)$
      $bsp\_send(map\_out_j)$
      $bsp\_send\_tag(tag\_out_j)$
      $bsp\_sync()$
IF    $task_j$ = "reduce"
THEN  $map\_out_j := bsp\_get()$
      $tag\_out_j := bsp\_get\_tag()$
      $reduce\_out_j := Agg(reduce\_out_j, map\_out_j, local\_tag_j, tag\_out_j)$
      $bsp\_sync()$
```

Inf-Ag-BSP Model

```
IF    $join_j$ = true
THEN  $join_j := Join\_Update()$
                  // Update the value of $join_i$ in current superstep
      IF    $join_j$ = true
            IF    $task_j$ = "map"
                  $map\_out_j := Map\_Function(\mathbf{s}_j)$
                  $tag\_out_j := Tag\_Fuction(\mathbf{s}_j)$
                  $bsp\_send(map\_out_j)$
                  $bsp\_send\_tag(tag\_out_j)$
                  $bsp\_sync()$
            IF    $task_j$ = "reduce"
                  $map\_out_j := bsp\_get()$
                  $tag\_out_j := bsp\_get\_tag()$
                  $reduce\_out_j := Aggre(reduce\_out_j, map\_out_j,$
                                          $local\_tag_j, tag\_out_j)$
                  $bsp\_sync()$
```

ZJUI

Next Steps

- What we have now is a computation model, however, to really achieve good performance in practice, we should develop a **scheduler** to assign work to active processors efficiently and effectively.

- The extended model, Inf-Ag-BSP model is not only bounded to implement MapReduce algorithm, its application on various fields can be investigated further.

- The API for extended BSP model should be developed as the general BSP model.

Conclusions

- ➢ MapReduce algorithm can be extended to handle stream, which is assumed to come indefinitely, via BSP model.
- ➢ For different classes of stream queries (memoryless, semi-memoryless and memorable), there are different MapReduce models to capture them.
- ➢ The requirement for large computing resources and large storage space incentives the extension of general BSP model:
  - BSP model gets extended to allow countably infinite number of agents in total, and each agent can join/leave the BSP computation.
  - A behavioral theory is developed on the basis of general BSP model.

Thanks for your attention!

Appendix: Proof for Characterization Theorem

- Definition of signature of ASM is consistent with signature restriction, it is satisfied trivially.
- Proof: Inf-Ag-BSP-ASM satisfies communication-and-joining postulate.

  (1) The cardinality of joining set $\mathcal{J}_n$ is finite;
  - Trivially satisfied since $\mathcal{J}$ is required to be always finite in the definition of $\mathcal{M}$.

  (2) If $i \notin \mathcal{J}_n$, then either $a_i \notin Ag_n$ or $(join_i, \mathbf{true}) \in \Delta_n$;
  - From Inf-Ag-BSP rule, if $\mathrm{val}(join_i) = \mathbf{false}$, the first two if-conditions of rule cannot be satisfied and third one must be satisfied. Then the machine either $skip$ (i.e. $i \notin \mathcal{J}_n$) or sets both $join_i$ and bar_i to $\mathbf{true}$ (thus $(join_i, \mathbf{true}) \in \Delta_n$).

  (3) & (4) conditions are related to barrier synchronization which are already proved in the characterization theorem of general BSP model's behavioral theorem.

  (5) If non-trivial update $(join_i, \mathbf{true}) \in \Delta_n$, then $(bar_i, \mathbf{true}) \in \Delta_n$;
  - Non-trivial update $(join_i, \mathbf{true})$ indicates that the initial value of $join_i$ is $\mathbf{false}$, so it corresponds to the join rule $r_{i,join}$. According to the definition of join rule, we must also have $bar_i := \mathbf{false} \in r_{i,join}$. Hence, $(bar_i, \mathbf{true}) \in \Delta_n$.

Appendix: Proof for Characterization Theorem (cont'd)

- Proof: Inf-Ag-BSP-ASM satisfies communication-and-joining postulate (cont'd)

  (6) If non-trivial update $(join_i, \mathbf{false}) \in \Delta_n$, then $\mathrm{val}_{S_{a_i(n)}}(bar_i) = \mathbf{false}$;

  - The update $(join_i, \mathbf{false}) \in \Delta_n$ must stem from a process rule, which can only be executed if $\mathrm{val}(bar_i) = \mathbf{false}$.

  (7) Whenever $(bar_i \wedge \neg barrier) \vee (\neg bar_i \wedge barrier)$ holds for any agent $i \in \mathcal{J}_n$ in state S_n, then $\Delta_{a_i(n)}\big(res(S_n, \Sigma_i)\big) = \emptyset$;

  - $(bar_i \wedge \neg barrier) \vee (\neg bar_i \wedge barrier)$ for $i \in \mathcal{J}_n$ is equivalent to $(join_i \wedge bar_i \wedge \neg barrier) \vee (join_i \wedge \neg bar_i \wedge barrier)$. In this case, none of the three if-conditions of Inf-Ag-BSP rule are satisfied, so ASM does nothing and the update set is empty.

  (8) The location $barrier$ is lazy, it only changes value when all bar_i for $i \in \mathcal{J}_n$ have changed their values from $\mathbf{true}$ to $\mathbf{false}$ (or $\mathbf{false}$ to $\mathbf{true}$ respectively).

  - According to the definition of SWITCH rule, $barrier$ gets changed only if all active agents have same values.

Appendix: Proof for Characterization Theorem (cont'd)

- Proof: Every Inf-Ag-BSP algorithm defined by the axiomatization can be simulated by an Inf-Ag-BSP-ASM.
  - For an Inf-Ag-BSP algorithm $\mathcal{B} = \{(a_i, \mathcal{A}_i | i \in \mathbb{N}\}$, we assume to have a concurrent ASM $\mathcal{M} = \{(a_i, \mathcal{M}_i | i \in \mathbb{N}\}$ with the same signature and concurrent runs.
  - For $\mathcal{M}_0$, its rule r_0 which is used to take care of the barrier synchronization and the joining behavior of Inf-Ag-BSP algorithm.
  - For other concurrent machines $\mathcal{M}_i$, their rules r_i have general form of:

$$\left(\text{IF } \varphi_1 \text{ THEN } r_{i,1}\right) | \ldots | \left(\text{IF } \varphi_m \text{ THEN } r_{i,m}\right)$$

  - Exploit the proof in general BSP mode gives that the general form can be written as the Inf-Ag-BSP rule with:

$$r_{i,proc} = \left(\text{IF } \varphi_1 \text{ THEN } r_{i,1}\right) | \ldots | \left(\text{IF } \varphi_{m'} \text{ THEN } r_{i,m'}\right)$$
$$r_{i,comm} = \left(\text{IF } \varphi_{m'+1} \text{ THEN } r_{i,m'+1}\right) | \ldots | \left(\text{IF } \varphi_{m''} \text{ THEN } r_{i,m''}\right)$$
$$r_{i,join} = \left(\text{IF } \varphi_{m''+1} \text{ THEN } r_{i,m''+1}\right) | \ldots | \left(\text{IF } \varphi_m \text{ THEN } r_{i,m}\right)$$