Pragel: A system for Large-Scale Graph Processing

Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski (Google – 2010)

Sualeh Asif

6.886 Talk
MIT Computer Science Artificial Intelligence Laboratory
Massachusetts Institute of Technology

March 30, 2021

PRAGEL :

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

GRAPH PROCESSING

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

Options:

(Large-scale)
GRAPH PROCESSING

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Options:

* Shared Memory : LIGRA, Julienne

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Options:

* Shared Memory: LIGRA, Julienne
  Boost(BGL), LEDA, NetworkX, JDSL, FGL ....

con: single computer

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Options:

* Shared Memory: LIGRA, Julienne, Boost(BGL), LEDA, NetworkX, JDSL, FGL ....
  - con: single computer
* Parallel / distributed systems: Parallel BGL, CGM graph (before PowerGraph) which we study next

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Options:

* Shared Memory : LIGRA, Julienne
  Boost(BGL), LEDA, NetworkX, JDSL, FGL ....
  con : single computer
* Parallel / distributed systems : Parallel BGL, CGM graph
  con : not good on fault-tolerence!

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Options:

* Shared Memory: LIGRA, Julienne, Boost(BGL), LEDA, NetworkX, JDSL, FGL ....
  - con: single computer
* Parallel / distributed systems: Parallel BGL, CGM graph
  - con: not good on fault-tolerence!
* Map Reduce: <no framework>
  - con: bad usability, bad performance, ill-suited for most graph algs.

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Options:

* Shared Memory: LIGRA, Julienne
  Boost(BGL), LEDA, NetworkX, JDSL, FGL ....
  con: single computer
* Parallel / distributed systems: Parallel BGL, CGM graph
  con: not good on fault-tolerence!
* Map Reduce : < no framework >
  con: bad usability, bad performance, ill-suited for most graph algs.
* Do it yourself .... ☹

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

RECAP:

(Large-scale) GRAPH PROCESSING

Solution:

PRAGEL

Options:

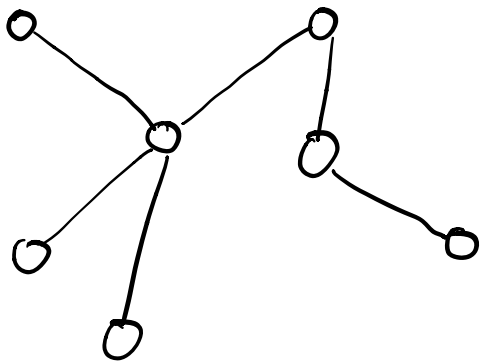
* Shared Memory: LIGRA, Julienne, Boost(BGL), LEDA, NetworkX, JDSL, FGL ....
  - con: single computer
* Parallel / distributed systems: Parallel BGL, CGM graph
  - con: not good on fault-tolerence!
* Map Reduce: <no framework>
  - con: bad usability, bad performance, ill-suited for most graph algs.
* Do it yourself .... :(

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL

PRAGEL



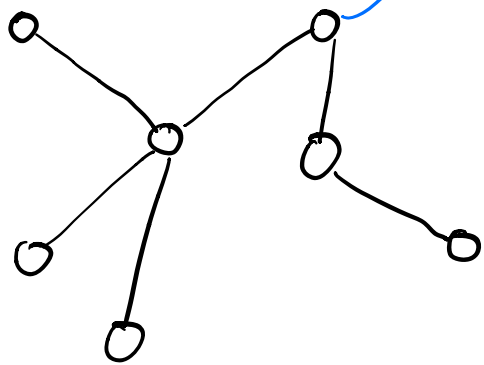
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL

PRAGEL

operate on all vertices in supersteps

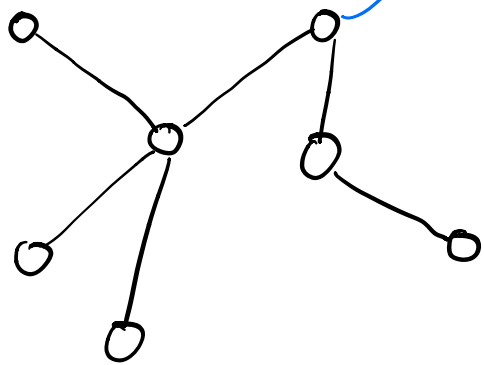


PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL

PRAGEL



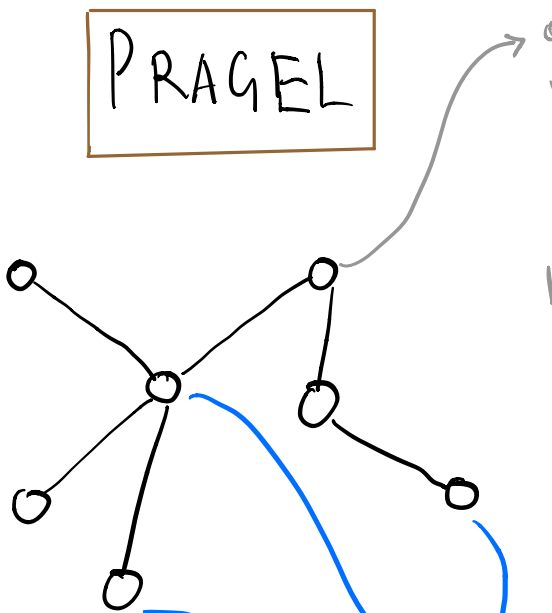
operate on all vertices in supersteps

In each superstep, operate on all vertices conceptually in parallel

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL



operate on all vertices in supersteps

In each superstep, operate on all vertices conceptually in parallel

Round S:
Every vertex, V, can
* read msg → V
* send msg → neighbor
* modify state

from S-1 ←
to S+1 ←

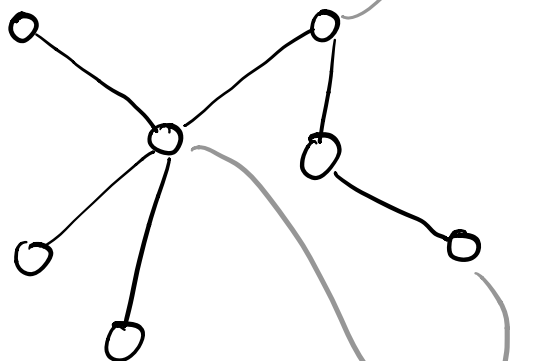
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

PRAGEL

* reminiscent of Map Reduce due to locality but tower (comm) overhead
* similar to Ligra in API for vertex.



operate on all vertices in supersteps

↓

In each superstep, operate on all vertices conceptually in parallel

↓

Round S:
Every vertex, V, can
* read msg → V
* send msg → neighbor
* modify state

from S-1 ← * read msg → V
to S+1 ← * send msg → neighbor
* modify state

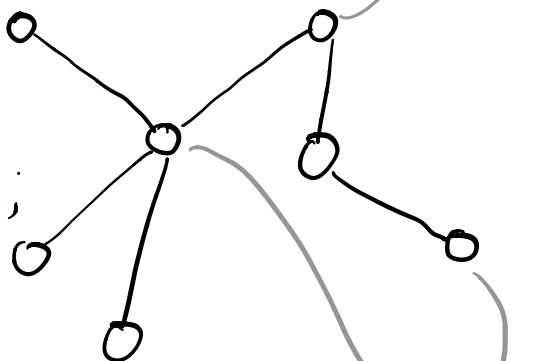
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

PRAGEL

* reminiscent of Map Reduce due to locality
* similar to Ligra in API for vertex.
* free of dead-locks; races by design.



operate on all vertices in supersteps

↓

In each superstep, operate on all vertices conceptually in parallel

↓

Round S:
Every vertex, V, can
* read msg → V
* send msg → neighbor
* modify state

from S-1 ← * read msg → V
to S+1 ← * send msg → neighbor
* modify state

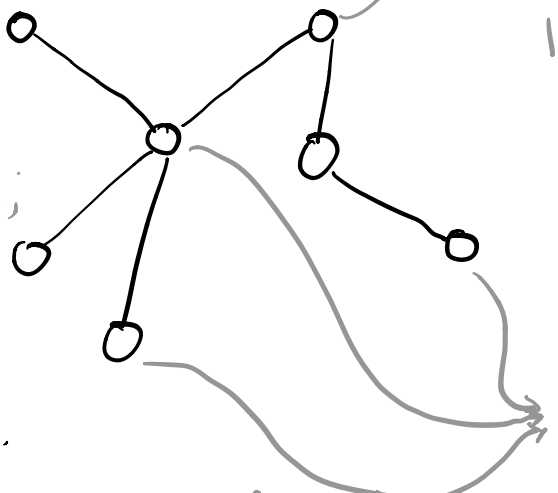
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

PRAGEL

* reminiscent of Map Reduce due to locality
* similar to Ligra in API for vertex.
* free of dead-locks; races by design.
* edges are not first-class citizens.



operate on all vertices in supersteps

↓

In each superstep, operate on all vertices conceptually in parallel

↓

Round S:
Every vertex, V, can
* read msg → V
* send msg → neighbor
* modify state

from S-1 ←
to S+1 ←

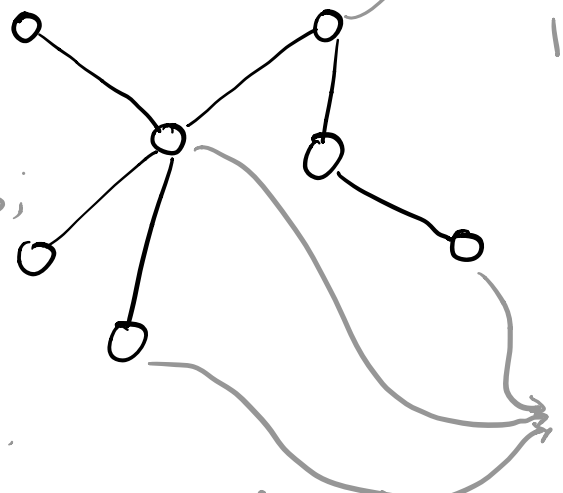
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

PRAGEL

* reminiscent of Map Reduce due to locality
* similar to Ligra in API for vertex.
* free of dead-locks, races by design.
* edges are not first-class citizens.
* pure message passing is performant and (sufficiently) expresive!



operate on all vertices in supersteps

↓

In each superstep, operate on all vertics conceptually in parallel

↓

Round S:
Every vertex, V, can
* read msg → V
* send msg → neighbor
* modify state

from S-1 ←
to S+1 ←

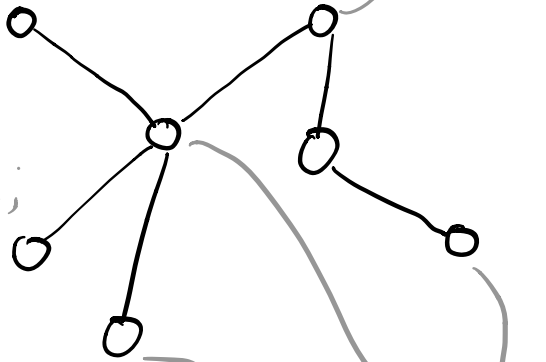
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

PRAGEL

* reminiscent of Map Reduce due to locality
* similar to Ligra in API for vertex.
* free of dead-locks, races by design.
* edges are not first-class citizens.
* pure message passing is performant and (sufficiently) expresive!



operate on all vertices in supersteps

In each superstep, operate on all vertices conceptually in parallel

Round S:
Every vertex, V, can
* read msg → V
* send msg → neighbor
* modify state

from S-1 ← V
to S+1 ← V

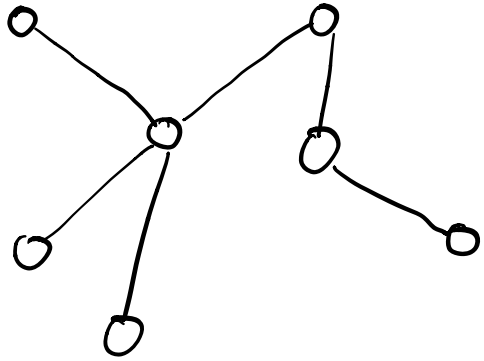
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

MORE DETAILS AND (C++) API

Model:
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in ngh(v)$
  * modify state



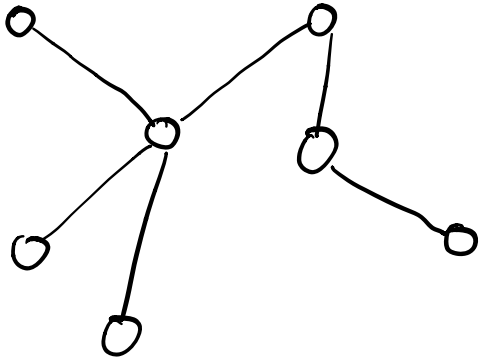
every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

MORE DETAILS AND (C++) API



Every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Model:
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in \text{nghn}(v)$
  * modify state

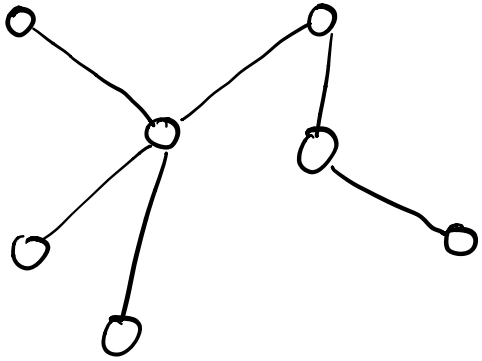
Halting:
* In each step, a vtx can vote to halt!

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

MORE DETAILS AND (C++) API



Every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Halting:
* In each step, a vtx can vote to halt!
* Program termintes when all vertices are halted!

Model:
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in \text{ngh}(v)$
  * modify state

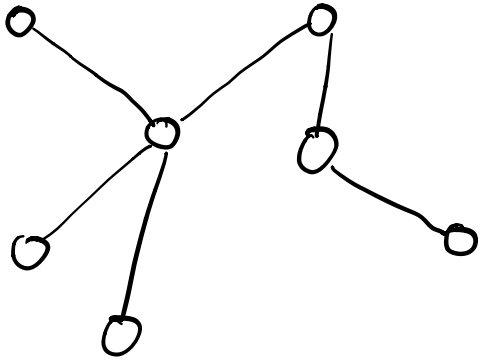
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

Model:
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghn(v)$
  * modify state

MORE DETAILS AND (C++) API



Halting:
* In each step, a vtx can vote to halt!
* Program termintes when all vertices are halted!
* vtx can be revived if it recieves a msg.

↳ every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

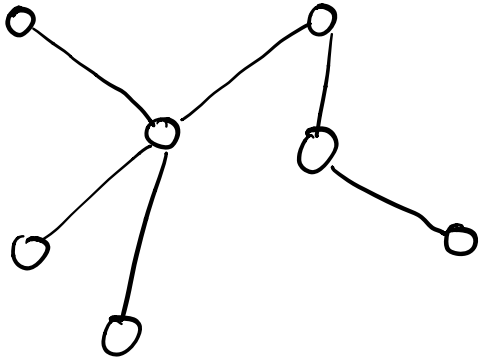
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

Model:
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in \text{ngh}(v)$
  * modify state

MORE DETAILS AND (C++) API



Halting:
* In each step, a vtx can vote to halt!
* Program termintes when all vertices are halted!
* vtx can be revived if it recieves msg.

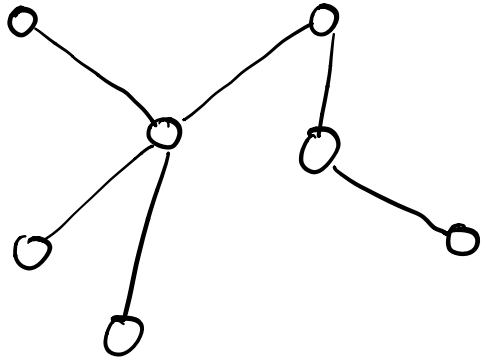
every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

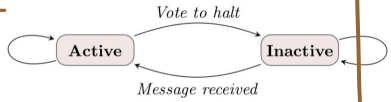
MODEL : message passing interface!

MORE DETAILS AND (C++) API



Every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Halting:



* In each step,
* Program can vote to halt when
  all vertices are halted!
* vtx can be revived if it recieves a msg.

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

Model:
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

PROCESSING

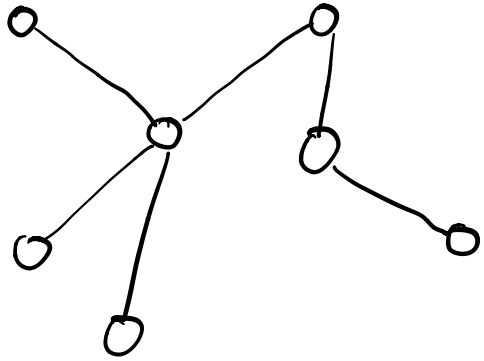
Model:
* itera
* operat

PRAGEL : DISTRIBUTED GRAPH

A SYSTEM FOR LARGE-SCALE GRAPH PR = .

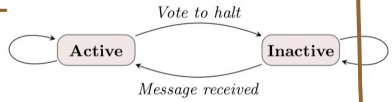
MODEL : message passing interface!

MORE DETAILS AND (C++) API



Every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Halting:



* In each step, a vtx can vote to halt!
* Progr
  all
* vtx am termintes when it vertices are halted!

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

templated ...

can be revived if recieves a msg.

virtual so user overrides

Model:

edge iterator

* itera
* operat

PROCESSING

te in supersteps
e on all vtx's (in parallel) in each step

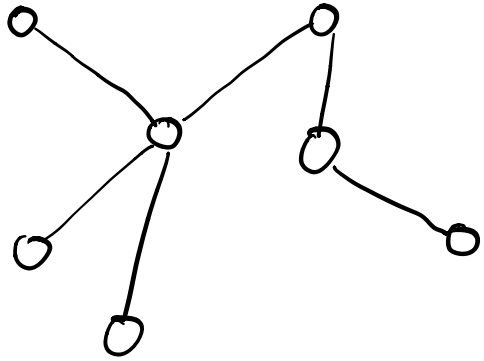
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

PRAGEL : DISTRIBUTED GRAPH

A SYSTEM FOR LARGE-SCALE GRAPH PR = .

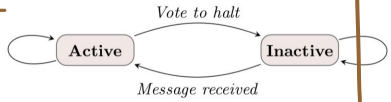
MODEL : message passing interface!

MORE DETAILS AND (C++) API



Every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Halting:



* In each step, a vtx can vote to halt!
* Progr all
* vtx am termintes when it vertices are halted!

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PROCESSING

te in supersteps
e on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

* first-class support for combiners & aggregators!

Model:
* itera
* operat

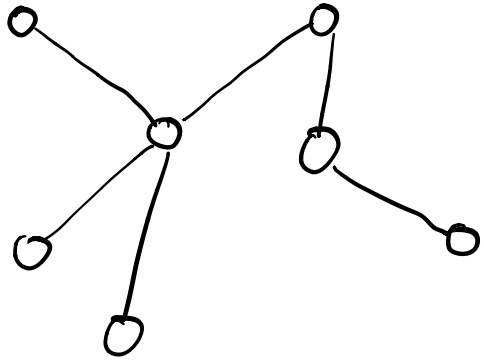
OCESSING

PRAGEL : DISTRIBUTED GRAPH

A SYSTEM FOR LARGE-SCALE GRAPH PR = .

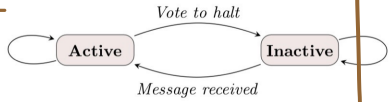
MODEL : message passing interface!

MORE DETAILS AND (C++) API



↳ every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Halting:



* In each step, a vtx can vote to halt!
* Progr all
* vtx am termintes when it vertices are halted!

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

can be revived if receives a msg.

PROCESSING

* te in supersteps
  e on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

* first-class support for combiners & aggregators!

very useful:
Δ-stepping via "min" eg aggregator!

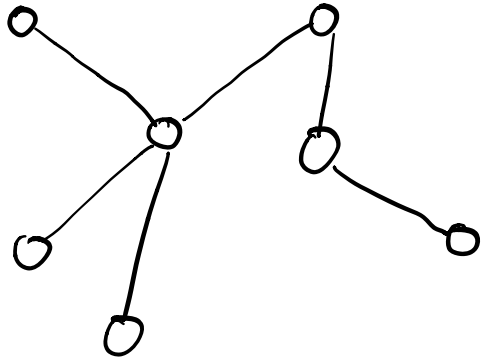
Model:
* itera
* operat

PRAGEL : DISTRIBUTED GRAPH

A SYSTEM FOR LARGE-SCALE GRAPH PR = .

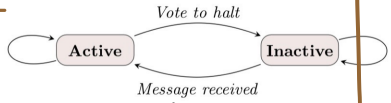
MODEL : message passing interface!

MORE DETAILS AND (C++) API



Every vtx has
* unique ID.
* mutable (user-defined) value.
* edge-list

Halting:



* In each step, a vtx can vote to halt!
* Progr all
* vtx am termintes when it vertices are halted!

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

can be revived if receives a msg.

PROCESSING

te in supersteps
e on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

* first-class support for combiners & aggregators!

very useful: Δ-stepping via "min" eg aggregator!

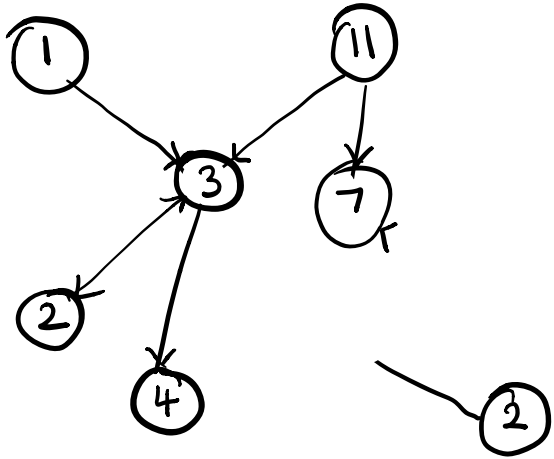
Model:
* itera
* operat

PRAGEL : DISTRIBUTED GRAPH

A SYSTEM FOR LARGE-SCALE GRAPH PR =

MODEL : message passing interface!

Example:



te in supersteps
e on all vtx's
(in parallel) in each
step

* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting:

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

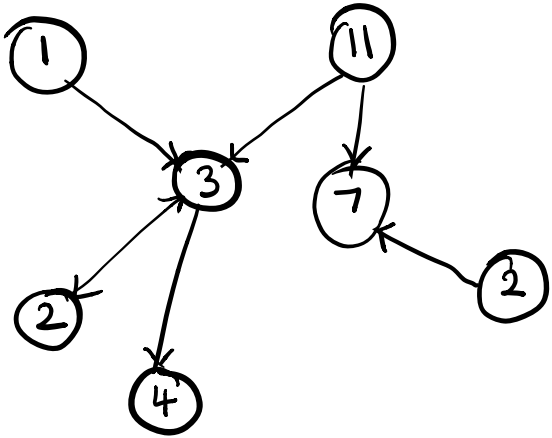
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

Example:

Superstep 0



Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghn(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

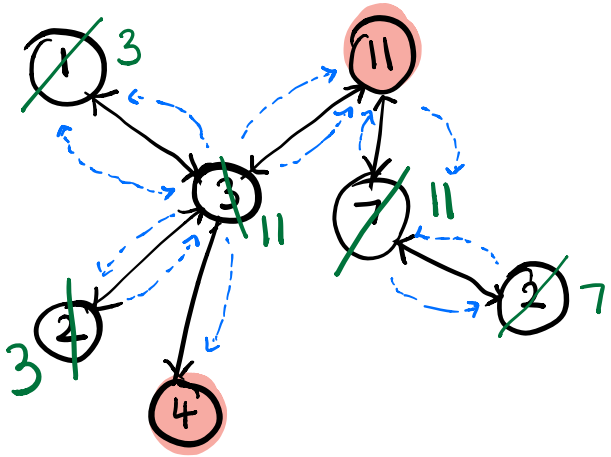
PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL: message passing interface!

Example:

Superstep 1



Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghn(v)$
  * modify state

Halting

C++ API: *Vote to halt*

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

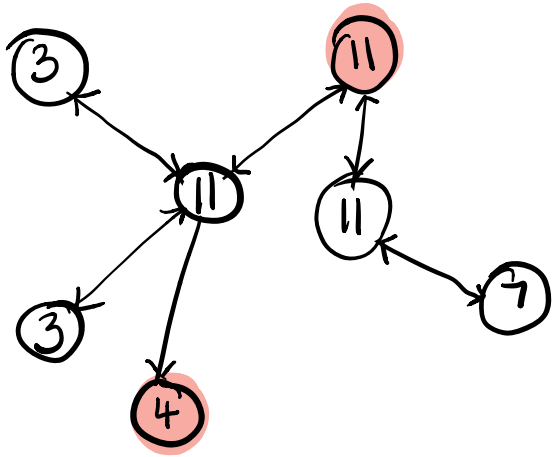
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

Example:

Superstep 1



Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghd(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

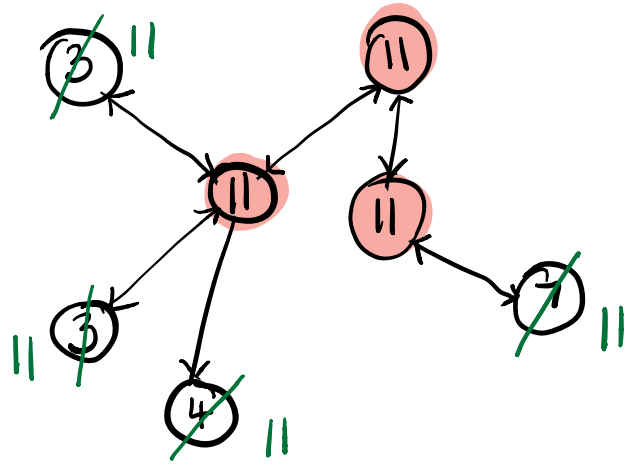
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

Example:

Superstep 2



Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghn(v)$
  * modify state

Halting

C++ API: *Vote to halt*

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

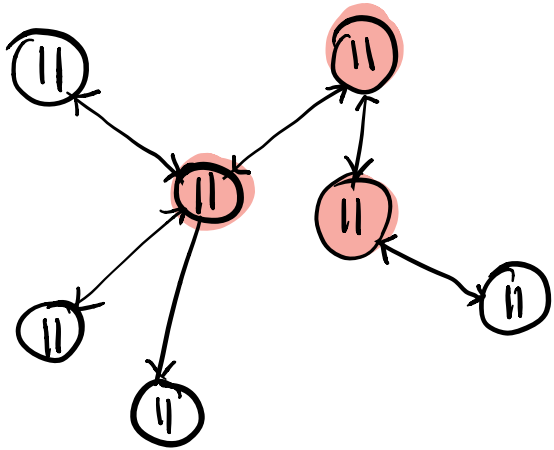
PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL: message passing interface!

Example:

Superstep 2



Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghd(v)$
  * modify state

Halting

C++ API: *Vote to halt*

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

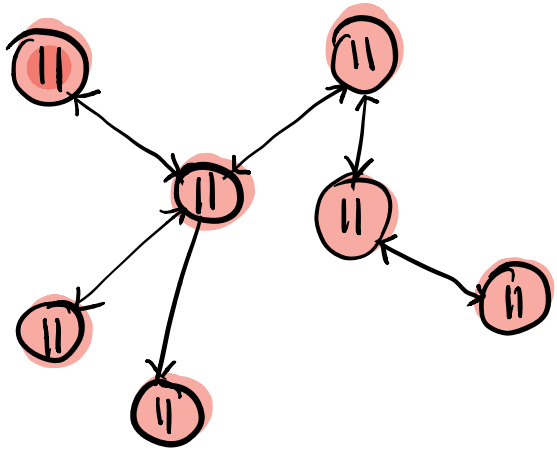
PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

MODEL : message passing interface!

Example:

Superstep 3



terminate!

Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in ngh(v)$
  * modify state

Halting

C++ API: *Vote to halt*

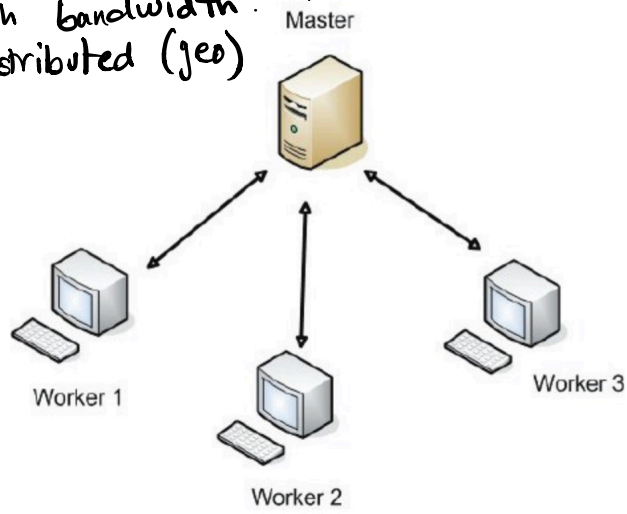
```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

IMPLEMENTATION:

* Google infra
  - thousands of PCs
  - high bandwidth
  - distributed (geo)



Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting

C++ API: *Vote to halt*

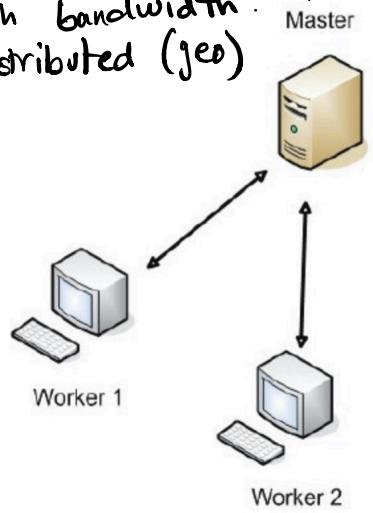
```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

IMPLEMENTATION:

* Google infra
  - thousands of PCs
  - high bandwidth
  - distributed (geo)



Details:

1) partitions graph (depending on ID)
   e.g. hash(ID) mod N

Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting

Vote to halt

C++ API:

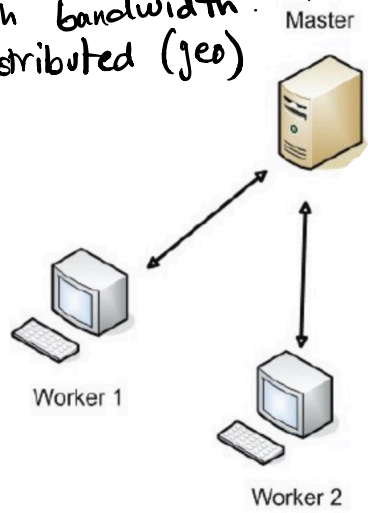
```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

IMPLEMENTATION:

* Google infra
  - thousands of PCs
  - high bandwidth
  - distributed (geo)



Details:

1) partitions graph (depending on ID)
   e.g. hash(ID) mod N
   can do custom allocation for locality

Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghd(v)$
  * modify state

Halting:

~~Vote to halt~~

C++ API:

Vote to halt

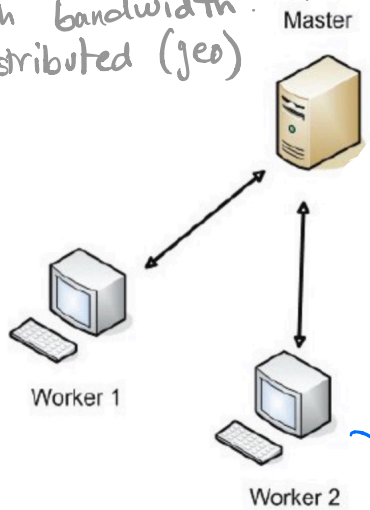
```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

IMPLEMENTATION:

* Google infra
  - thousands of PCs
  - high bandwidth
  - distributed (geo)



Details:

1) partitions graph (depending on ID)
   e.g. hash(ID) mod N
   can do custom allocation for locality

2) **Master**:
   assigned nothing

   Workers:
   - assigned partition by master.
   - maintains its portion of state
   - enques msgs.

Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting: Vote to halt

C++ API:

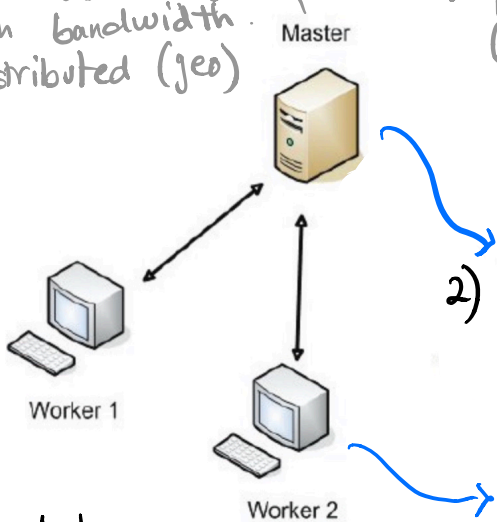
```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

IMPLEMENTATION:

* Google infra
  - thousands of PCs
  - high bandwidth
  - distributed (geo)



Details:

1) partitions graph (depending on ID)
   e.g. hash(ID) mod N
   can do custom allocation for locality

2) Master :
   - assigned nothing

   Workers :
   - assigned partition by master.
   - maintains its partion of state
   - enques msgs.

* fault-tolerance
  - check-pointing by each worker. saves full state!!
  - if fail: master reassigns... computation loaded and reated.

Mode :
* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S :
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting

C++ API:

Vote to halt

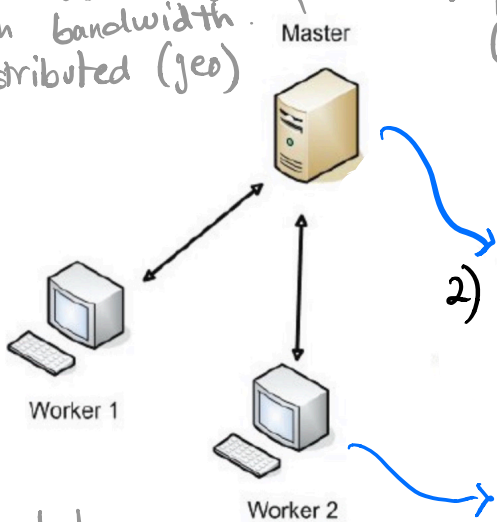
```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

IMPLEMENTATION:

* Google infra
  - thousands of PCs
  - high bandwidth
  - distributed (geo)



* fault-tolerance
  - check-pointing by each worker. saves full state!!
  - if fail: master reassigns computation loaded and repeated.

Details:

1) partitions graph (depending on ID)
   e.g. hash(ID) mod N
   can do custom allocation for locality

2) Master :
   - assigned nothing

   Workers :
   - assigned partition by master.
   - maintains its partion of state
   - enques msgs.

Mode :

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S :
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghr(v)$
  * modify state

Halting: Vote to halt

C++ API:

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS!: ← finally!

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghn(v)$
  * modify state

Halting

C++ API: *Vote to halt*

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS!:

PAGE RANK:

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in \text{nghr}(v)$
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

CATIONS!:

APPLI RANK:

PAGE

inherits from Vertex

vertex edge message

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ nghn(v)
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

CATIONS!

APPLI RANK:

PAGE

inherits from Vertex

vertex edge message

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

on step

Pog

Compute each s

erank update.

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS:

PAGE RANK:

inherits from Vertex

vertex edge message

```
class PageRankVertex
    : public Vertex<double, void, double> {
public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```

on step

Pog

compute each s

rank update.

rounds

message passing

halt after 30 rounds.

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghn(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS:

SSSP

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

Model:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghd(v)$
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS!:

SSSP:

vtx value: init +INF

EdgeValue: (weighted)

Vertex recieves potential min distances from source!

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghr(v)$
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS!:

SSSP:

vtx value: init +INF

EdgeValue: (weighted)

Vertex recieves potential min distances from source!

vtx updates value.

saves min value

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghr(v)$
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS!:

SSSP:

vtx value: init +INF

EdgeValue: (weighted)

Vertex recieves potential min distances from source!

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

vtx updates value.

saves min value

sends updated min dist to neighbors

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in \text{ngh}(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS!:

SSSP:

vtx value: init +INF

EdgeValue: (weighted)

Vertex recieves potential min distances from source!

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
void Compute(MessageIterator* msgs) {
  int mindist = IsSource(vertex_id()) ? 0 : INF;
  for (; !msgs->Done(); msgs->Next())
    mindist = min(mindist, msgs->Value());
  if (mindist < GetValue()) {
    *MutableValue() = mindist;
    OutEdgeIterator iter = GetOutEdgeIterator();
    for (; !iter.Done(); iter.Next())
      SendMessageTo(iter.Target(),
                    mindist + iter.GetValue());
  }
  VoteToHalt();
}
};
```

vtx updates value.

saved val

saves min value

sends updated min dist to neighbors

halt (may be revived later.)

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghr(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

APPLICATIONS! :

SSSP:

vtx value : init +INF
EdgeValue : (weighted)

Vertex recieves potential min distances from source!

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
void Compute(MessageIterator* msgs) {
  int mindist = IsSource(vertex_id()) ? 0 : INF;
  for (; !msgs->Done(); msgs->Next())
    mindist = min(mindist, msgs->Value());
  if (mindist < GetValue()) {
    *MutableValue() = mindist;
    OutEdgeIterator iter = GetOutEdgeIterator();
    for (; !iter.Done(); iter.Next())
      SendMessageTo(iter.Target(),
                    mindist + iter.GetValue());
  }
  VoteToHalt();
}
};
```

vtx updates value.

saves min value

sends updated min dist to neighbors

halt (may be revived later.)

can be replaced by combiner

```
class MinIntCombiner : public Combiner<int> {
  virtual void Combine(MessageIterator* msgs) {
    int mindist = INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    Output("combined_source", mindist);
  }
};
```

Mode :

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S :
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghr(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

EXPERIMENTS :

* 300 PC's

SSSP:

Binary trees:

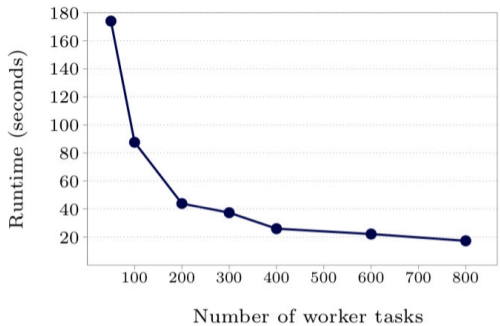


Figure 7: SSSP—1 billion vertex binary tree: varying number of worker tasks scheduled on 300 multi-core machines

Mode :

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S :
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

~~EXPERI~~ MLNLS:

~~SSSP:~~

Binary trees:

* 300 PCs

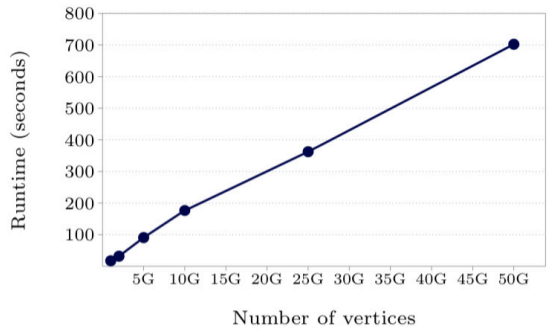


Figure 8: SSSP—binary trees: varying graph sizes on 800 worker tasks scheduled on 300 multicore machines

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → $w \in nghr(v)$
  * modify state

Halting

C++ API:

Vote to halt

```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

PRAGEL: DISTRIBUTED GRAPH PROCESSING

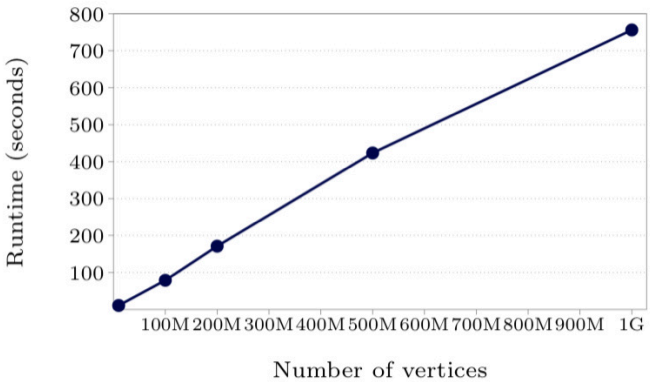
A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

EXPERIMENTS:

* 300 PCs

SSSP:

Log-normal random graphs:



Warning: implementation is not optimized!
- to show satisfactory perf with clean impl.

Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ nghr(v)
  * modify state

Halting:

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```

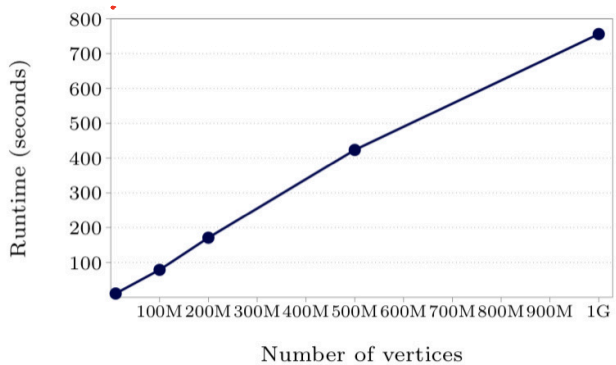
PREGEL : DISTRIBUTED GRAPH PROCESSING

A SYSTEM FOR LARGE-SCALE GRAPH PROCESSING.

THANK YOU! - QUESTIONS?

```
class ShortestPathVertex
    : public Vertex<int, int, int> {
  void Compute(MessageIterator* msgs) {
    int mindist = IsSource(vertex_id()) ? 0 : INF;
    for (; !msgs->Done(); msgs->Next())
      mindist = min(mindist, msgs->Value());
    if (mindist < GetValue()) {
      *MutableValue() = mindist;
      OutEdgeIterator iter = GetOutEdgeIterator();
      for (; !iter.Done(); iter.Next())
        SendMessageTo(iter.Target(),
                      mindist + iter.GetValue());
    }
    VoteToHalt();
  }
};
```

```
class PageRankVertex
    : public Vertex<double, void, double> {
 public:
  virtual void Compute(MessageIterator* msgs) {
    if (superstep() >= 1) {
      double sum = 0;
      for (; !msgs->Done(); msgs->Next())
        sum += msgs->Value();
      *MutableValue() =
          0.15 / NumVertices() + 0.85 * sum;
    }

    if (superstep() < 30) {
      const int64 n = GetOutEdgeIterator().size();
      SendMessageToAllNeighbors(GetValue() / n);
    } else {
      VoteToHalt();
    }
  }
};
```



Mode:

* iterate in supersteps
* operate on all vtx's (in parallel) in each step
* Round S:
  Every vertex, V, can
  * read msg → V
  * send msg → w ∈ ngh(v)
  * modify state

Halting

C++ API:


```
template <typename VertexValue,
          typename EdgeValue,
          typename MessageValue>
class Vertex {
 public:
  virtual void Compute(MessageIterator* msgs) = 0;

  const string& vertex_id() const;
  int64 superstep() const;

  const VertexValue& GetValue();
  VertexValue* MutableValue();
  OutEdgeIterator GetOutEdgeIterator();

  void SendMessageTo(const string& dest_vertex,
                     const MessageValue& message);
  void VoteToHalt();
};
```