IEEE/ACM CCGRID 2024
The 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

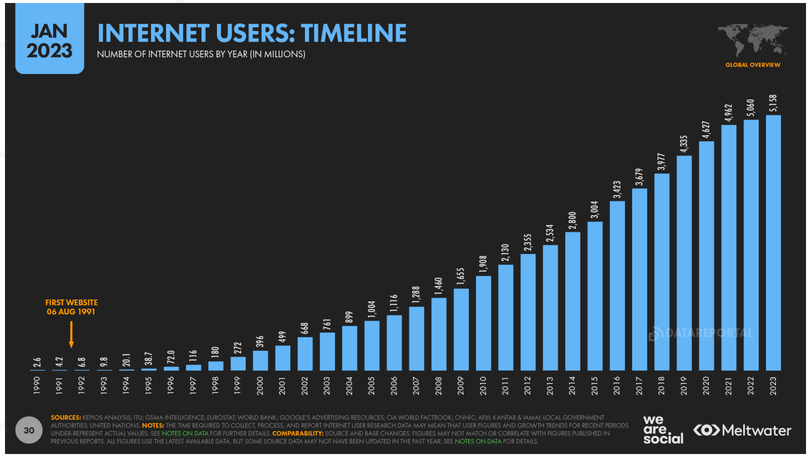
A Distributed, Asynchronous Algorithm for Large-Scale Internet Network Topology Analysis

Youssef Elmougy*, Akihiro Hayashi, and Vivek Sarkar

Habanero Extreme Scale Software Research Lab
Georgia Institute of **Tech**nology

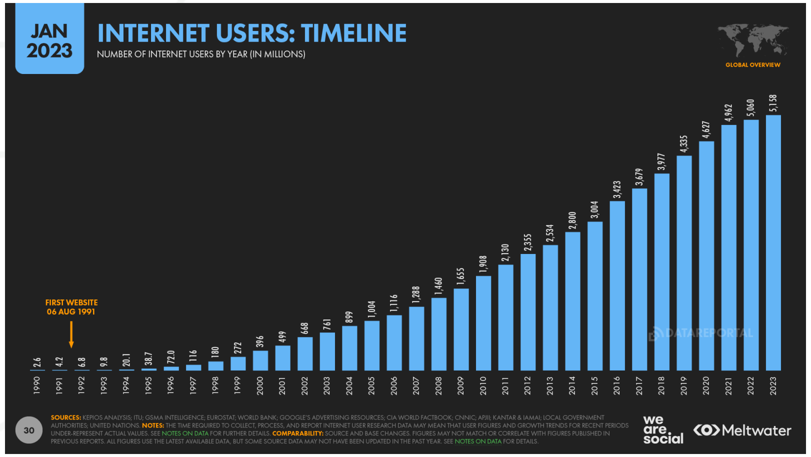
* Corresponding Author and Presenting Author: yelmougy3@gatech.edu

The Growth of the Internet and the increasing complexity of its global network topology

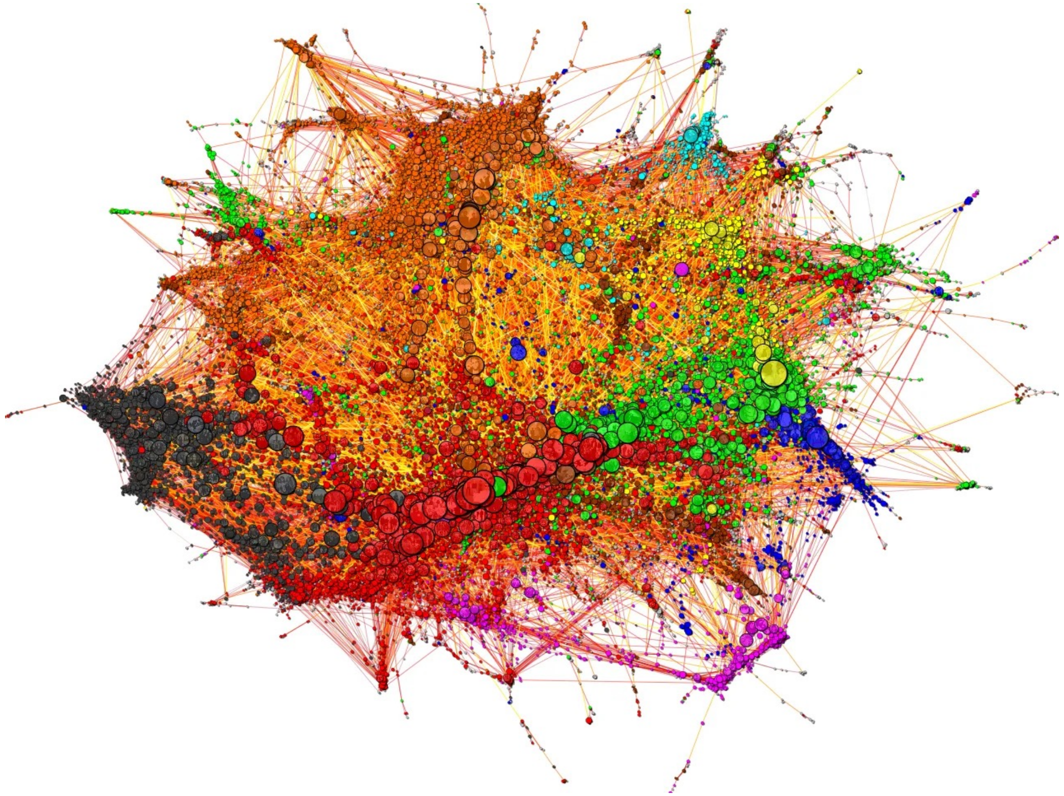


Picture borrowed from:
https://datareportal.com/reports/digital-2023-global-overview-report

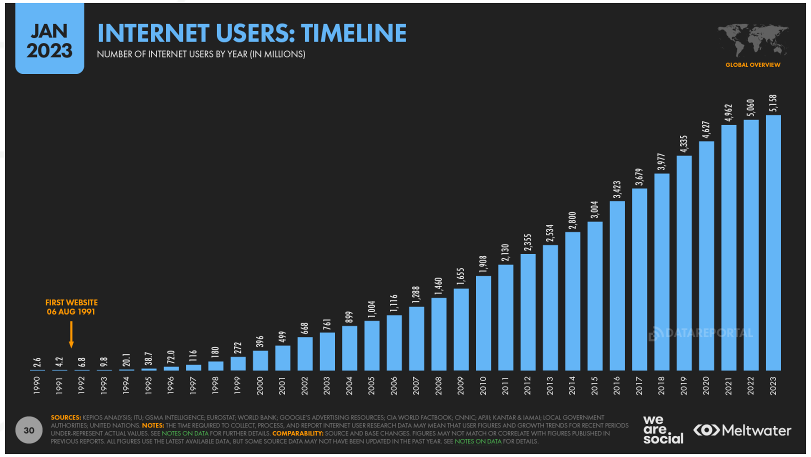
The Growth of the Internet and the increasing complexity of its global network topology



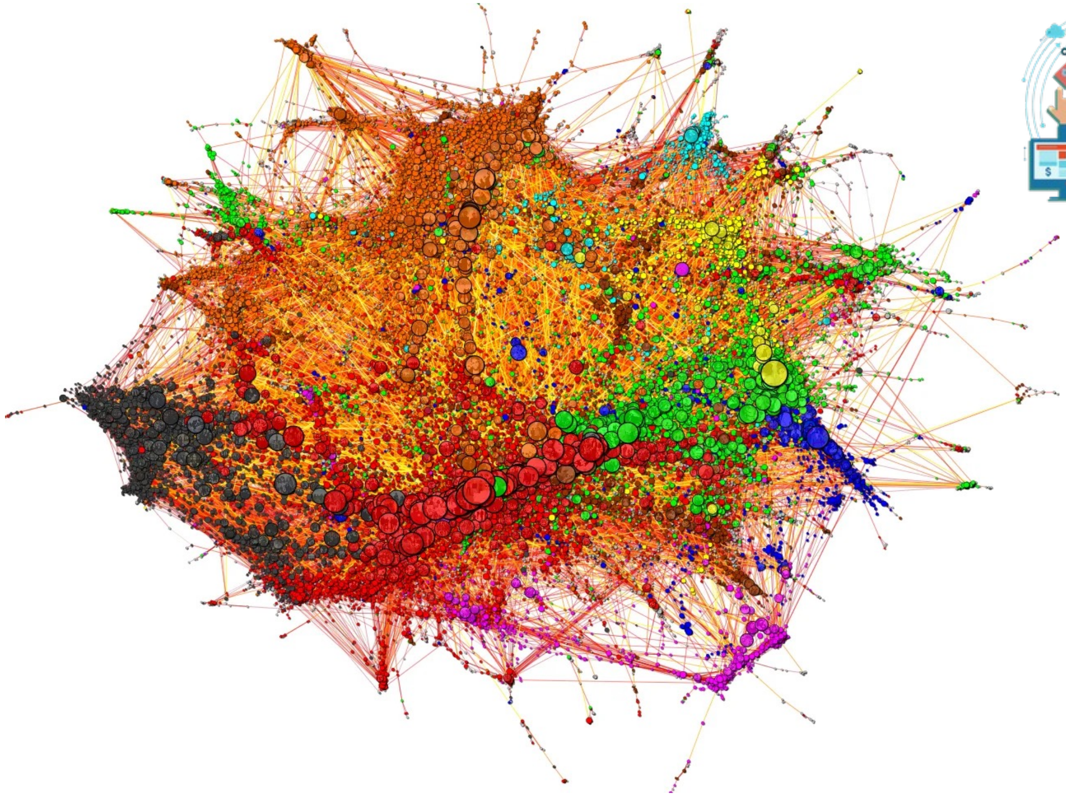
Picture borrowed from:
https://datareportal.com/reports/digital-2023-global-overview-report

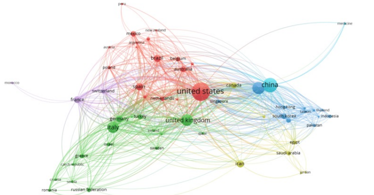


The Growth of the Internet and the increasing complexity of its global network topology

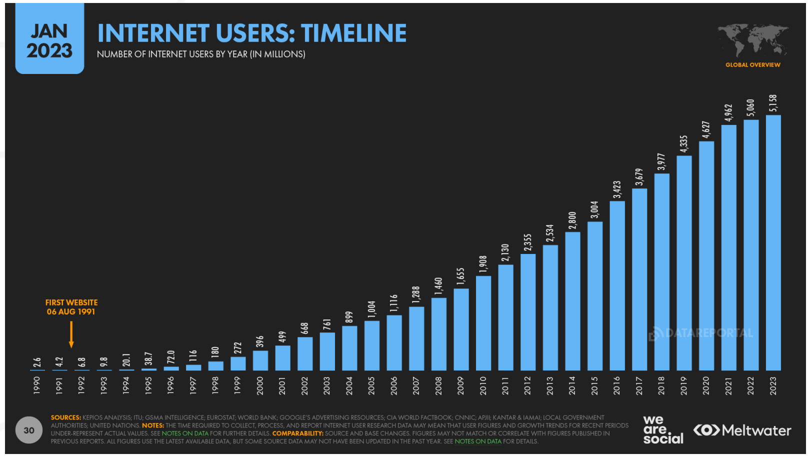


Picture borrowed from:
https://datareportal.com/reports/digital-2023-global-overview-report

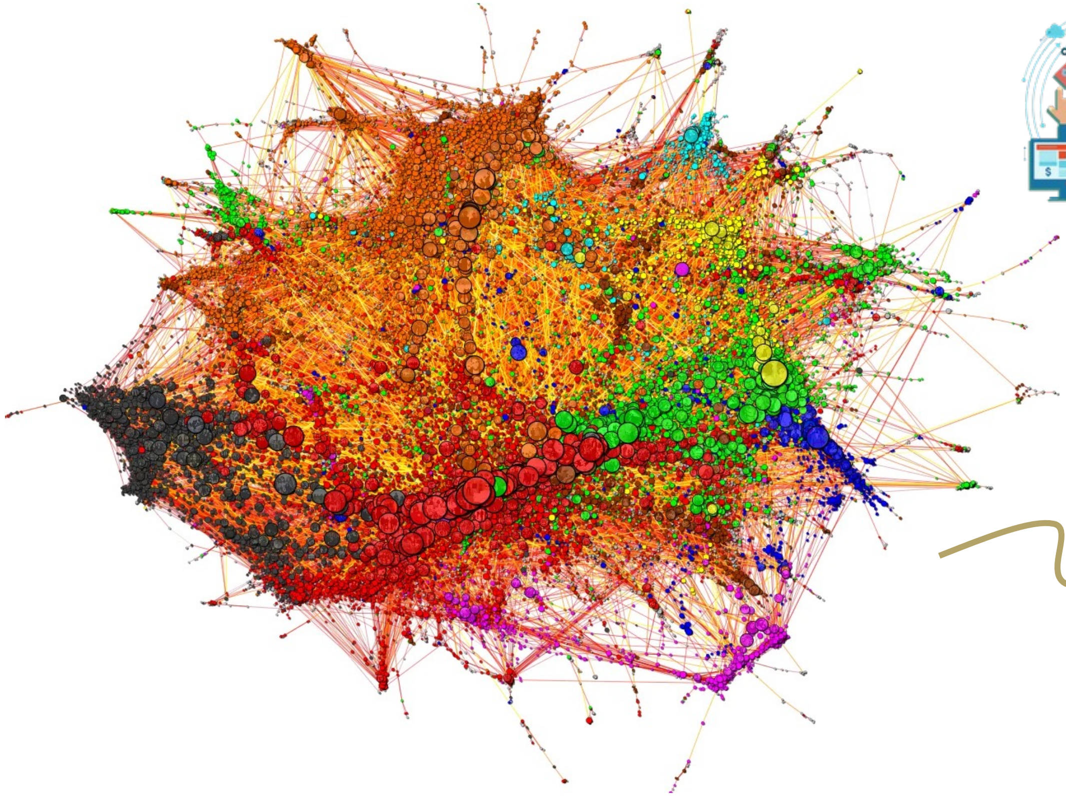


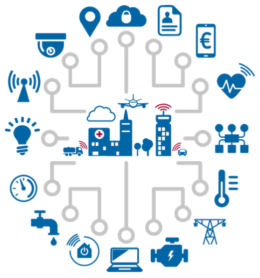


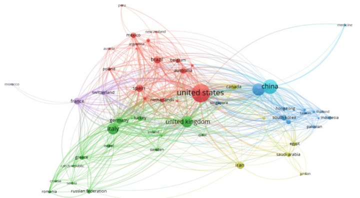
The Growth of the Internet and the increasing complexity of its global network topology



Picture borrowed from:
https://datareportal.com/reports/digital-2023-global-overview-report

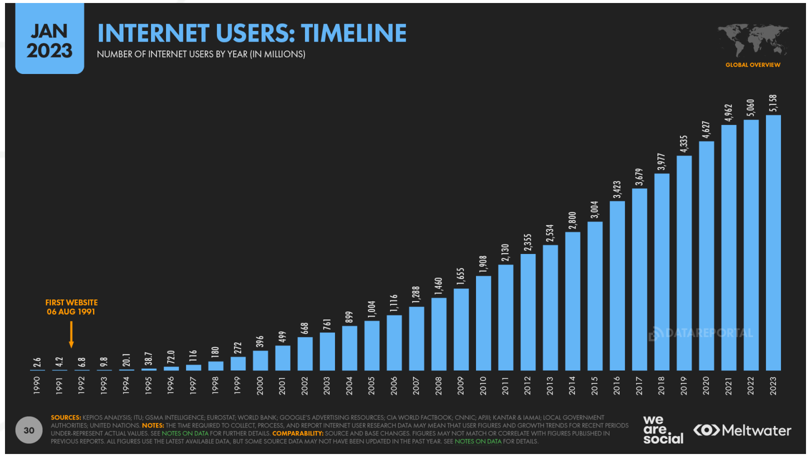




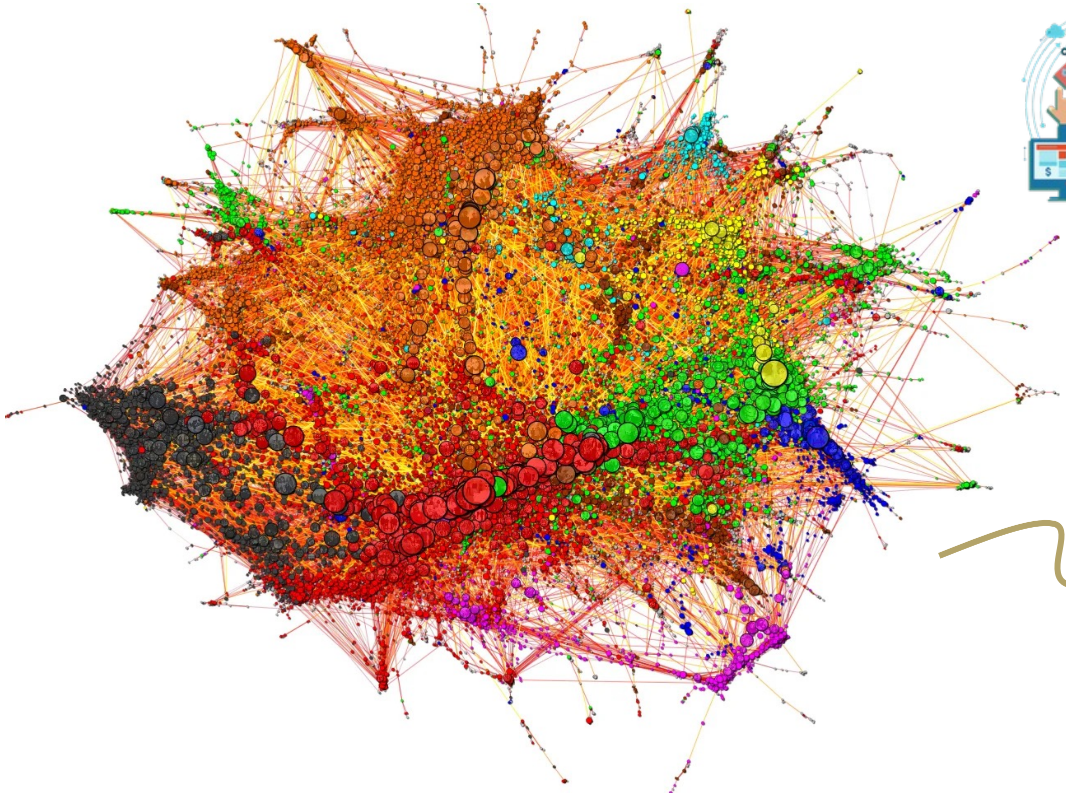


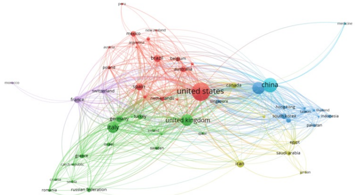
Advanced analytic tools and algorithms capable of determining the centrality and importance of network topology have become increasingly important

The Growth of the Internet and the increasing complexity of its global network topology



Picture borrowed from:
https://datareportal.com/reports/digital-2023-global-overview-report





Advanced analytic tools and algorithms capable of determining the centrality and importance of network topology have become increasingly important

Effectively understanding and analyzing the critical nodes and components within these networks is paramount to:

Network Management

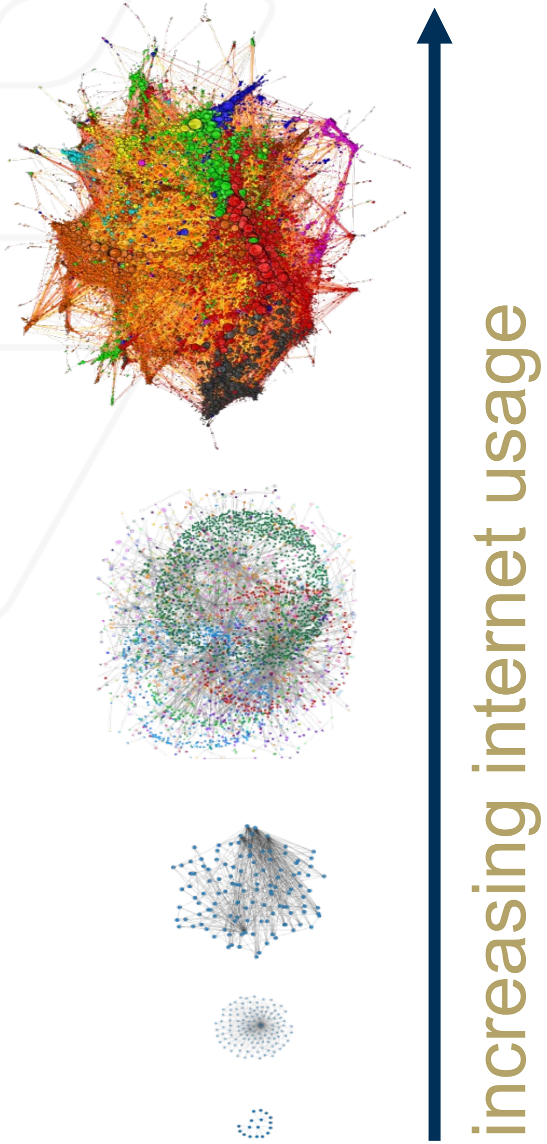
Security

Fault Tolerance and Resilience

Network Stability and Performance

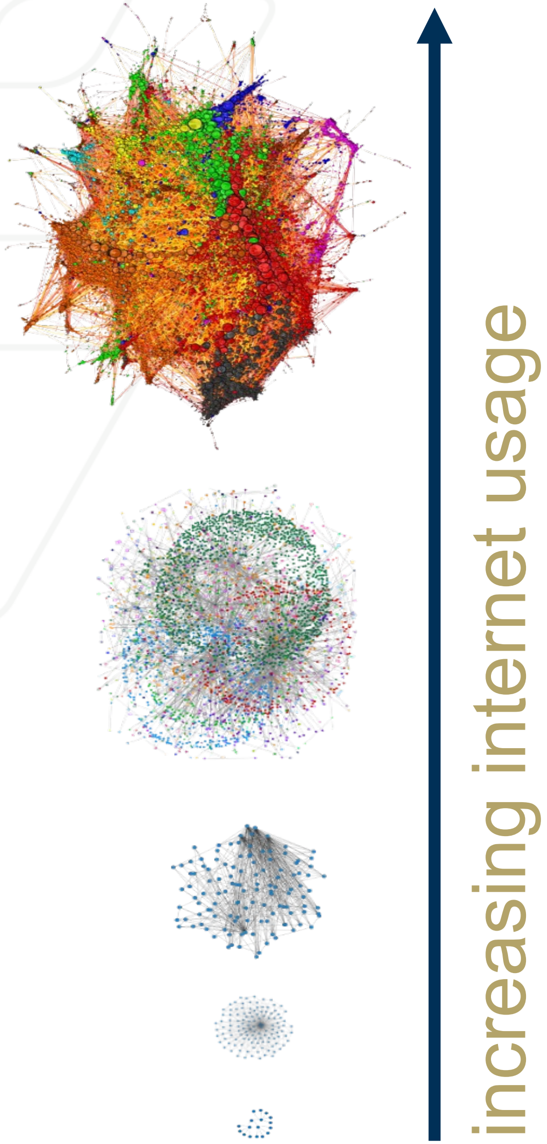
Georgia Tech

The Scalability Problem



- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem



Individual User

- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem

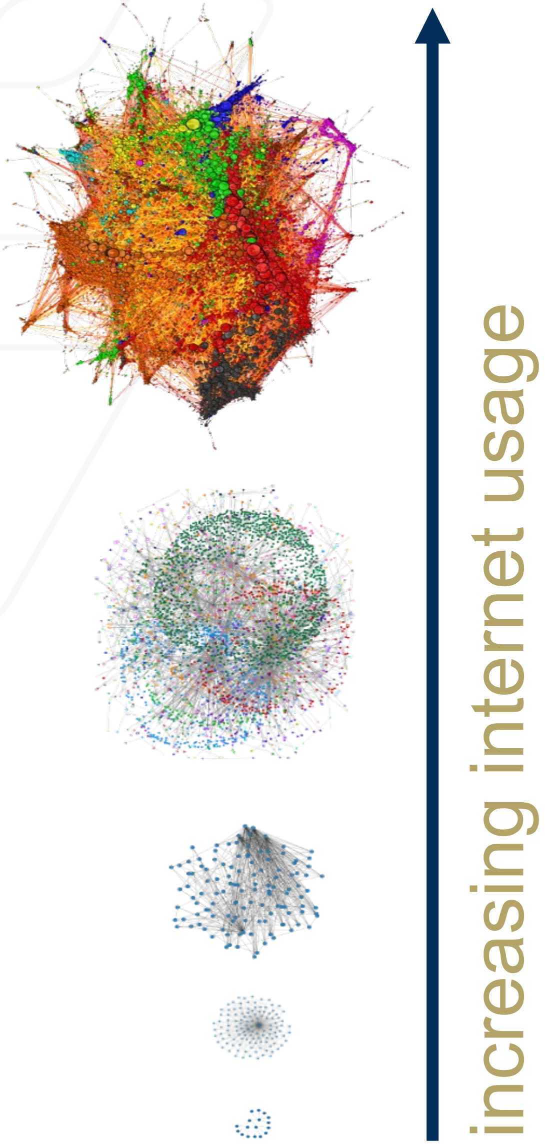
increasing internet usage

Individual User

Massive Corporations

- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem



Individual User

Massive Corporations

Modern Datacenter

with hundreds of thousands of servers

- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem

increasing internet usage

Individual User

Massive Corporations

Modern Datacenter

with hundreds of thousands of servers

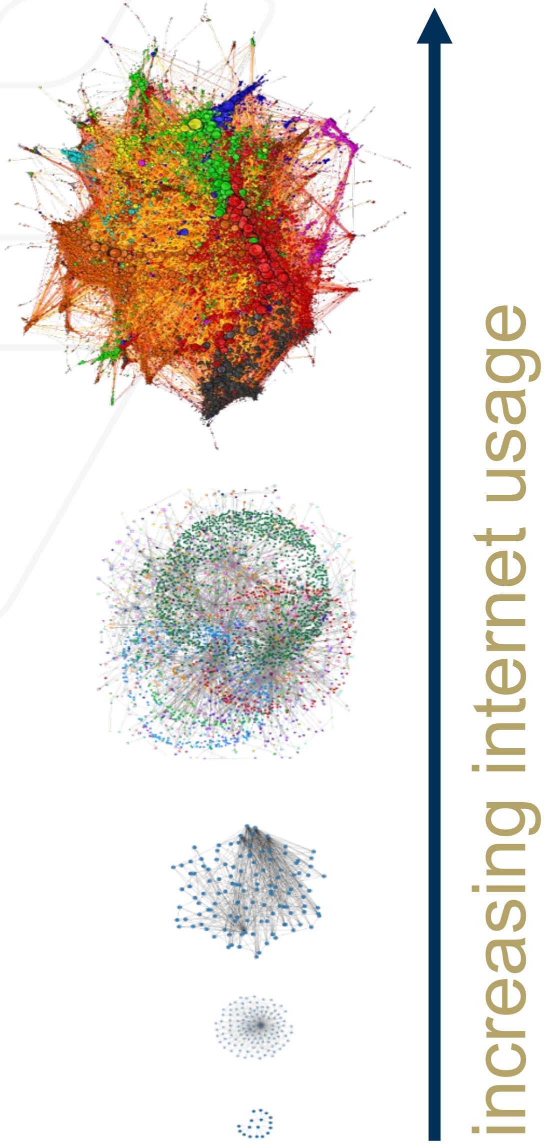
DATACENTERS

growing increasingly at an exponential rate

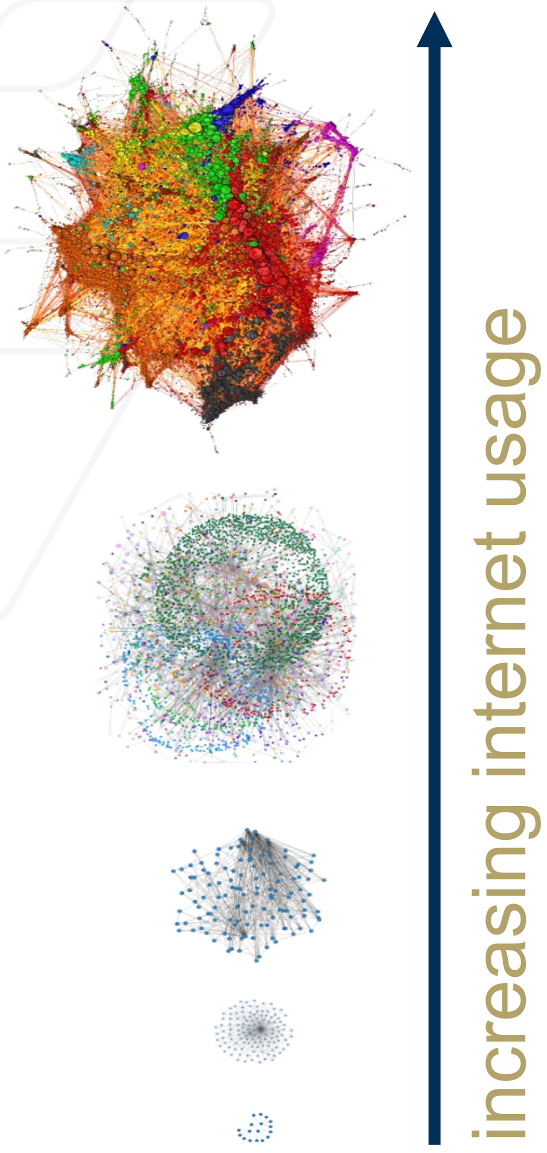
- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

Georgia Tech

The Scalability Problem



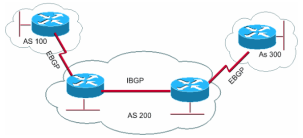
The Scalability Problem




Routers

Bridges and Switches

127.0.0.1
172.16.0.9
192.0.0.7
IPs


ASes


Repeaters and Hubs


NICs

BILLIONS OF DEVICES

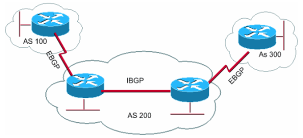
The Scalability Problem

Routers

Bridges and Switches

127.0.0.1
172.16.0.9
192.0.0.7

IPs



ASes

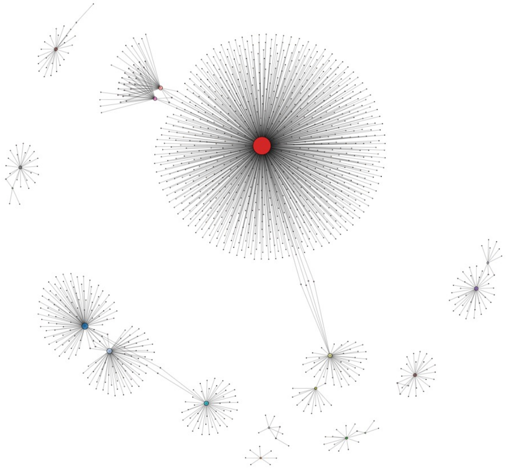
Repeaters and Hubs

NICs

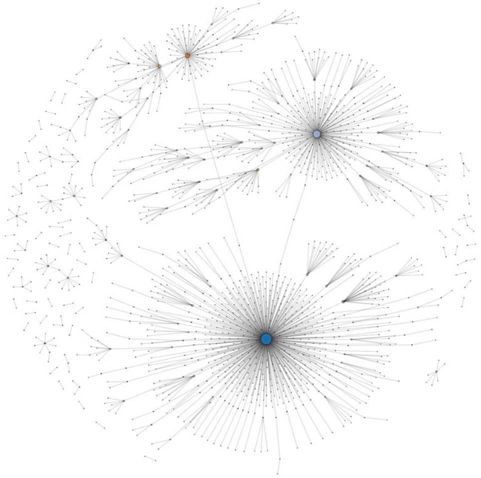
BILLIONS OF DEVICES

LARGER MAGNITUDE OF CONTINUOUS INTERACTIONS

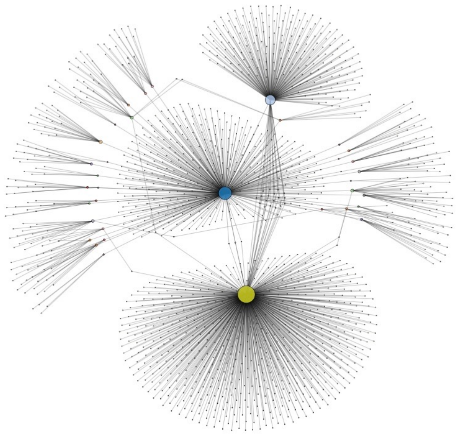
increasing internet usage



(a) Exchanges across Routers



(b) Exchanges across IPs



(c) Exchanges across ASes

Georgia Tech

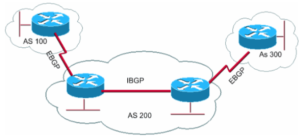
The Scalability Problem

Routers

Bridges and Switches

127.0.0.1
172.16.0.9
192.0.0.7

IPs

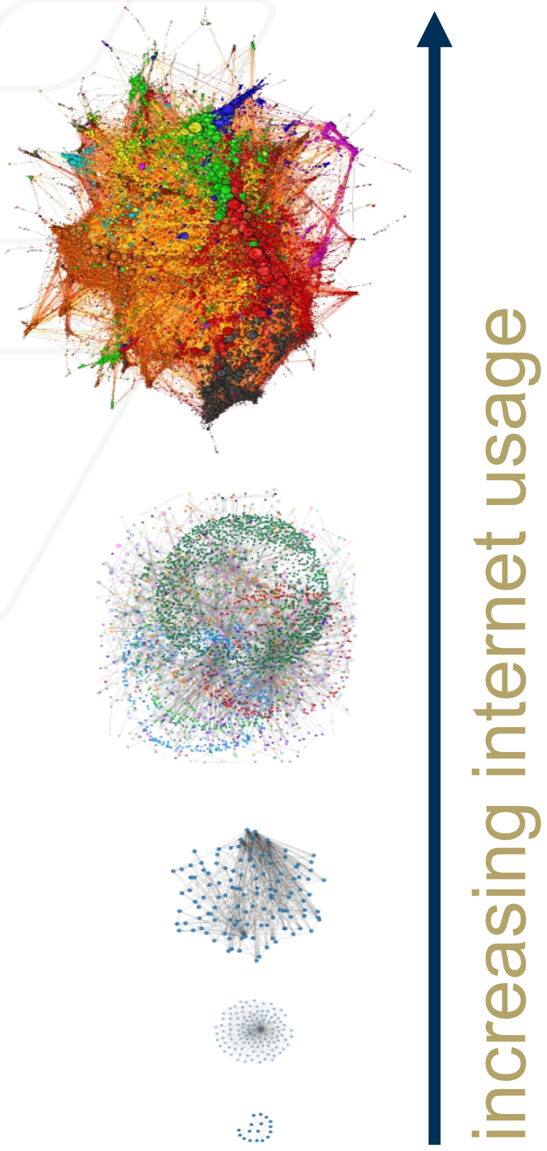


ASes

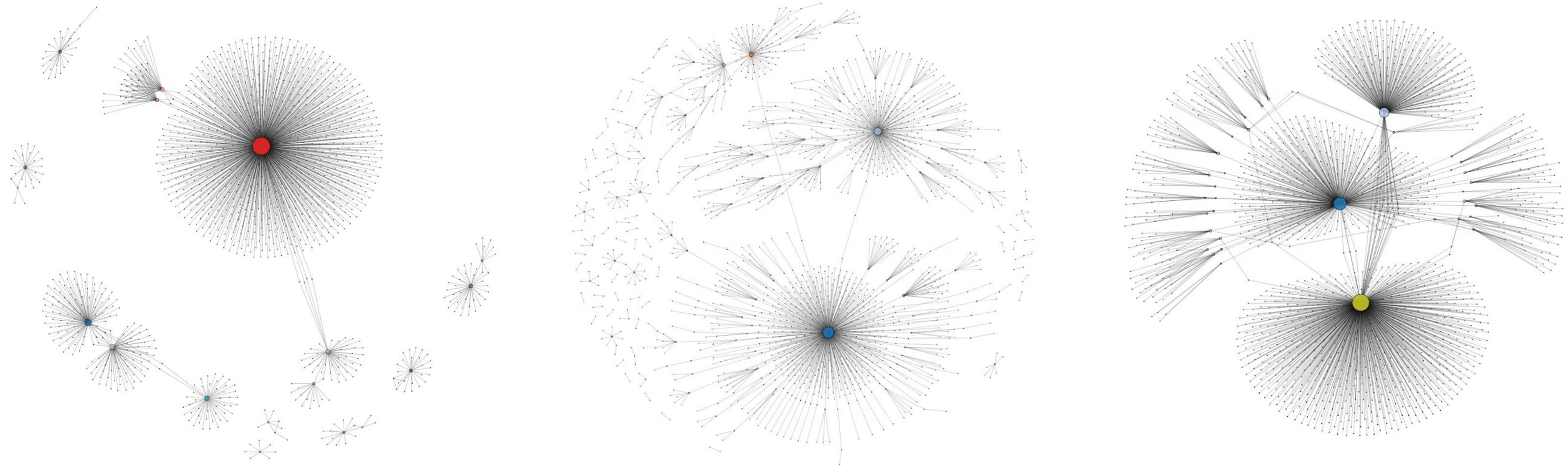
BILLIONS OF DEVICES

Repeaters and Hubs

NICs



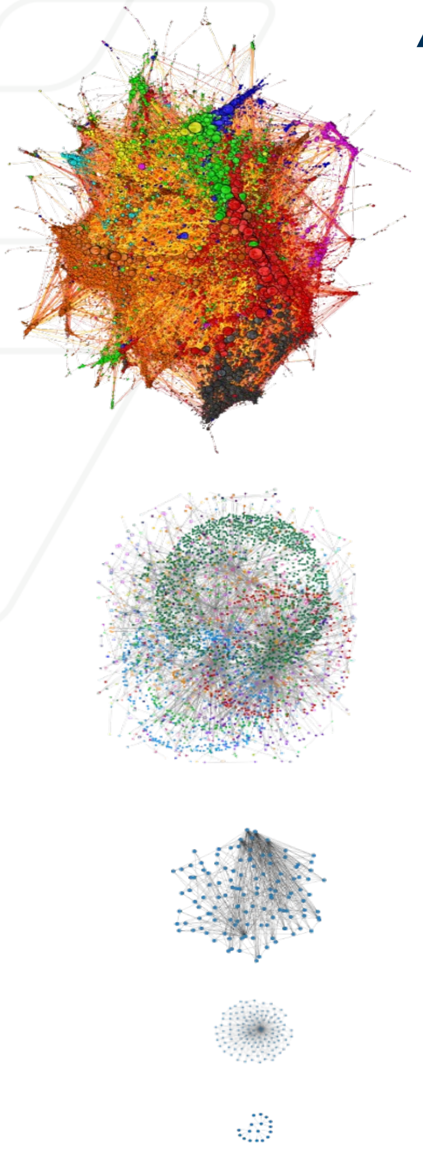
LARGER MAGNITUDE OF CONTINUOUS INTERACTIONS



(a) Exchanges across Routers (b) Exchanges across IPs (c) Exchanges across ASes

- There is a need for parallel computing resources to meet the computational and memory requirements

The Scalability Problem

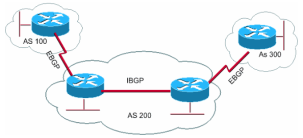


increasing internet usage

Routers

Bridges and Switches

127.0.0.1
172.16.0.9
192.0.0.7
IPs



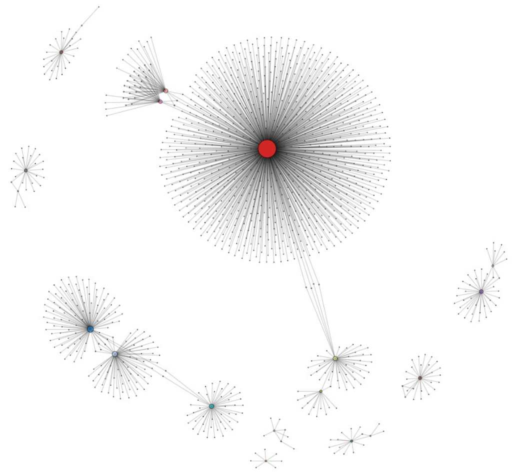
ASes

BILLIONS OF DEVICES

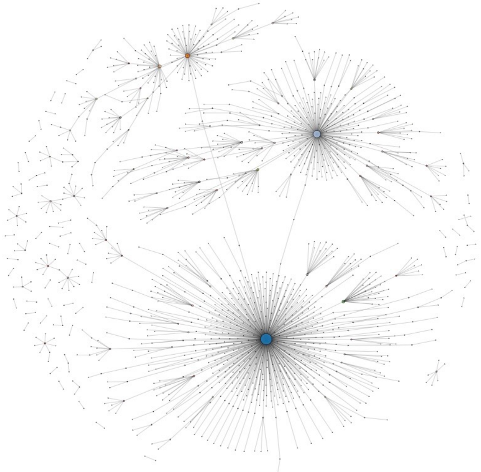
Repeaters and Hubs

NICs

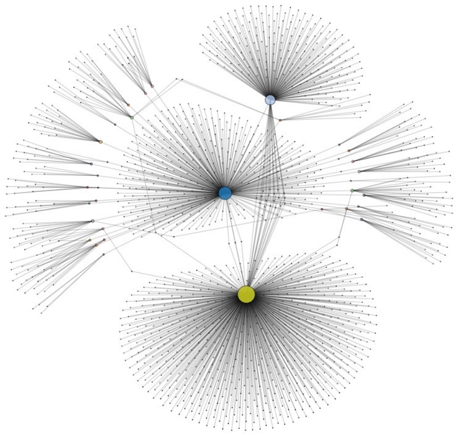
LARGER MAGNITUDE OF CONTINUOUS INTERACTIONS



(a) Exchanges across Routers



(b) Exchanges across IPs



(c) Exchanges across ASes

- ❑ There is a need for parallel computing resources to meet the computational and memory requirements

It is important to develop scalable and lightweight systems to efficiently process large-scale network topologies!

Georgia Tech

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities**, **inefficiencies**, and **possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities**, **inefficiencies**, and **possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

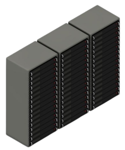
CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities**, **inefficiencies**, and **possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g. Microsoft Azure



1M servers

200 datacenters

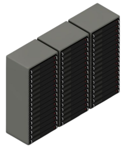
Georgia Tech

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities**, **inefficiencies**, and **possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g. Microsoft Azure



1M servers

200 datacenters

which manages **diverse arrays of clients with unique agreements and connectivity requirements**

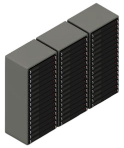
Georgia Tech

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities**, **inefficiencies**, and **possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

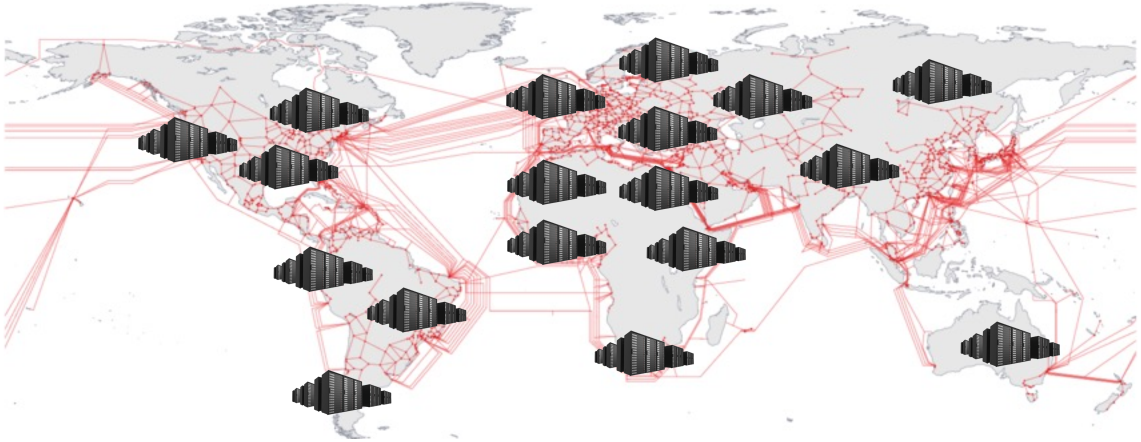
CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g. Microsoft Azure



1M servers

200 datacenters



which manages **diverse arrays of clients with unique agreements and connectivity requirements**

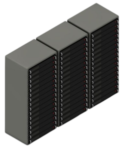
operates **across several datacenters in multiple continents** relying on **networks of routers, switches, and computing devices**

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities**, **inefficiencies**, and **possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g. Microsoft Azure

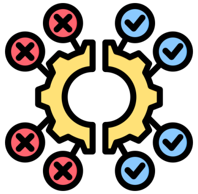


1M servers

200 datacenters

operates **across several datacenters in multiple continents** relying on **networks of routers, switches, and computing devices**

which manages **diverse arrays of clients with unique agreements and connectivity requirements**



DISRUPTIONS

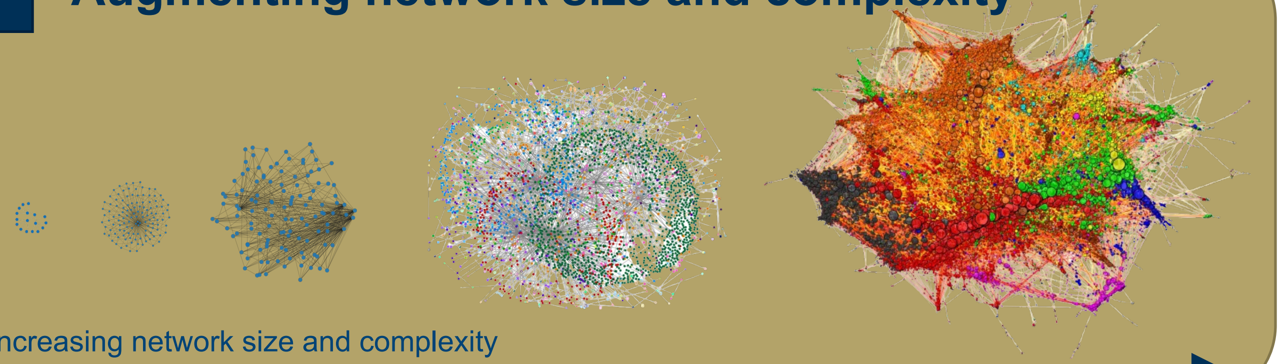
as a result of **performance bottlenecks**, **component failures**, or **security breaches** can have dire consequences

The 4 Challenges in Reliable Internet Provision

Georgia Tech

The 4 Challenges in Reliable Internet Provision

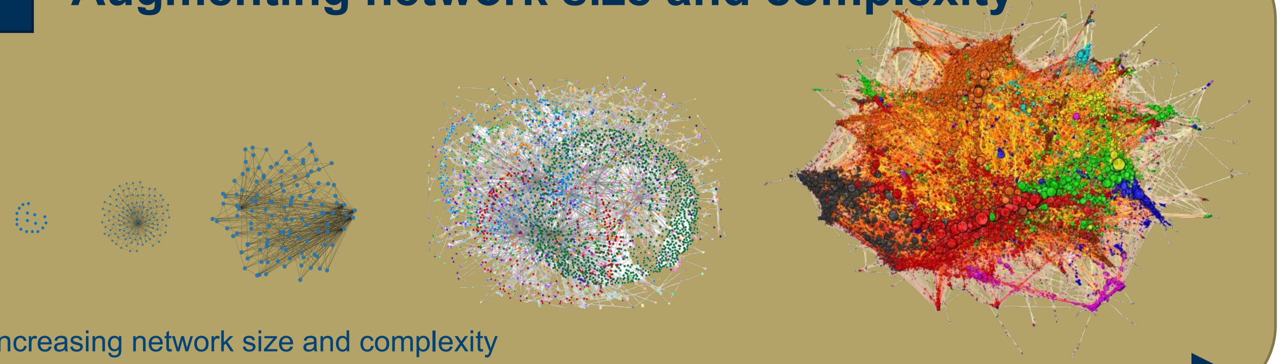
1 **Augmenting network size and complexity**



increasing network size and complexity

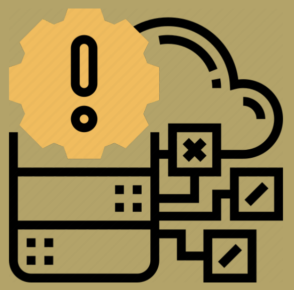
The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity



increasing network size and complexity

2 Fault tolerance and security



The 4 Challenges in Reliable Internet Provision

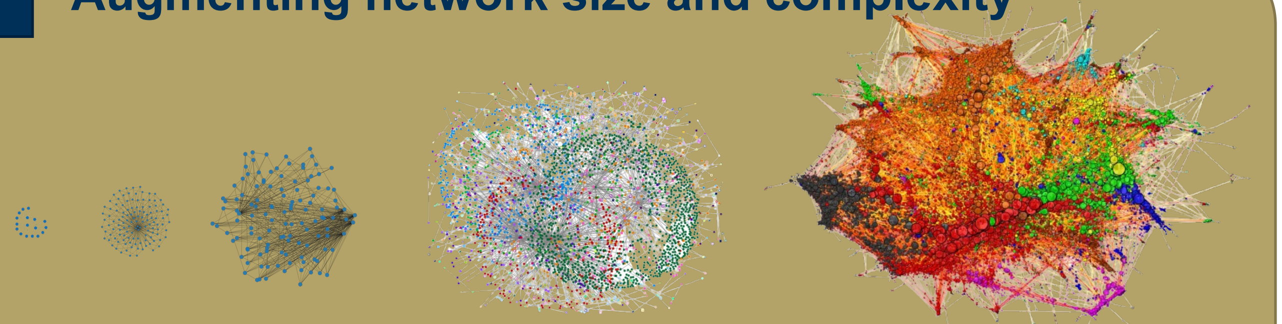
1 Augmenting network size and complexity

3 Real-time decision making

2 Fault tolerance and security

The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity



increasing network size and complexity

3 Real-time decision making

2 Fault tolerance and security

4 Energy considerations

The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity

3 Real-time decision making

2 Fault tolerance and security

4 Energy considerations

This paper studies the scalability of our novel algorithm to overcome the inherent challenges of internet provision by performing large-scale network topology analysis!

Georgia Tech

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] ...

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] **...**

are designed and optimized for **smaller scale networks and graphs**

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] ...

are designed and optimized for **smaller scale networks and graphs**

are limited in scalability due to their application of **sequential or inefficient algorithms**

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] **...**

are designed and optimized for **smaller scale networks and graphs**

are limited in scalability due to their application of **sequential or inefficient algorithms**

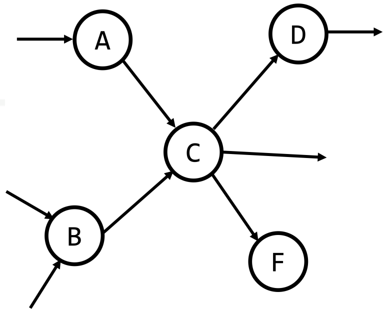
CHALLENGE

Makes it difficult to process the expansive and intricate dynamic web of interconnected devices constituting the modern internet

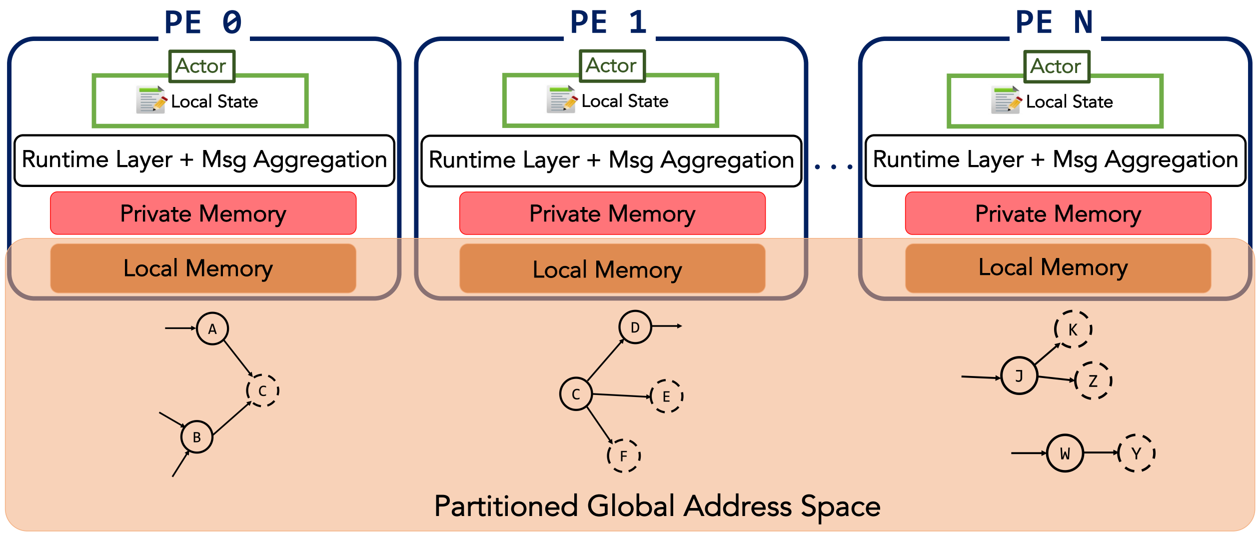
Georgia Tech

Backend Actor-based Scalable Architecture

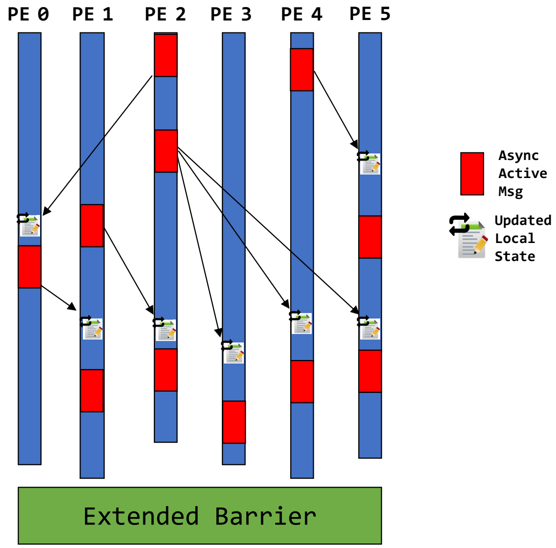
Sample Graph



Distributed Actor Runtime



Execution Model



- Presents a lightweight, asynchronous computation model
- Utilizes fine-grained asynchronous actor messages to express point-to-point remote operations
- Treats actors as primitives of computation, where actors are inherently isolated and share no mutable state
- Actors process messages sequentially within its mailbox, thereby avoiding data races and synchronization

NOTE: “Actor” and “Selector” will be used interchangeably

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:
finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:
finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

Georgia Tech

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

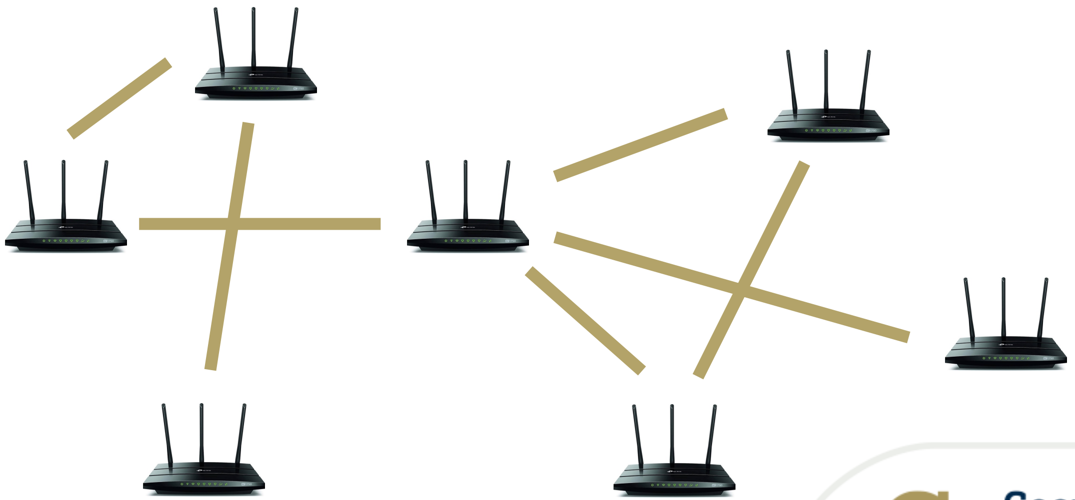
Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:
finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

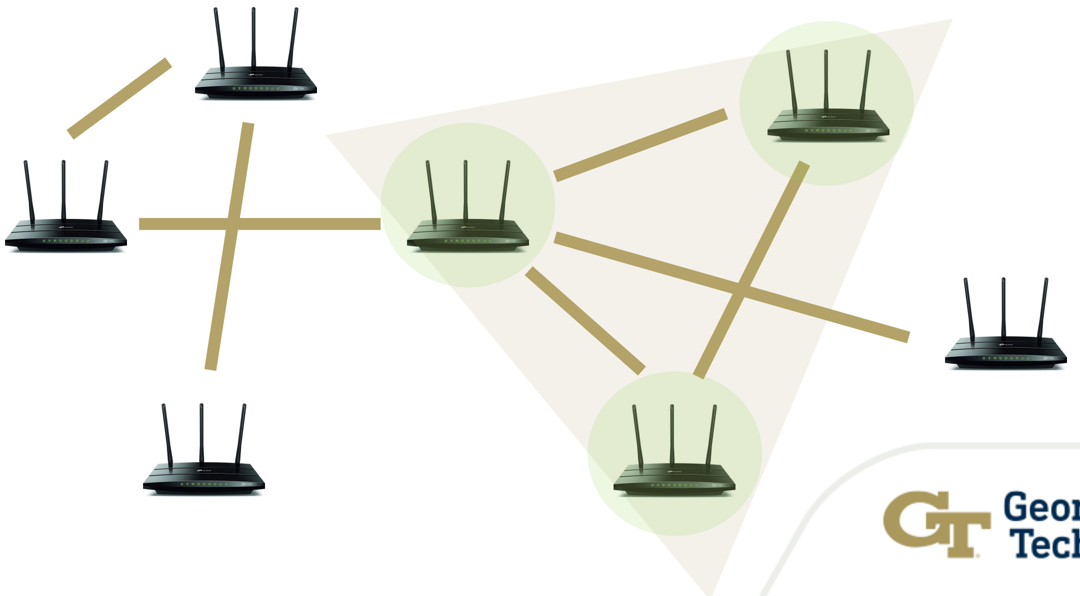
Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:
finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

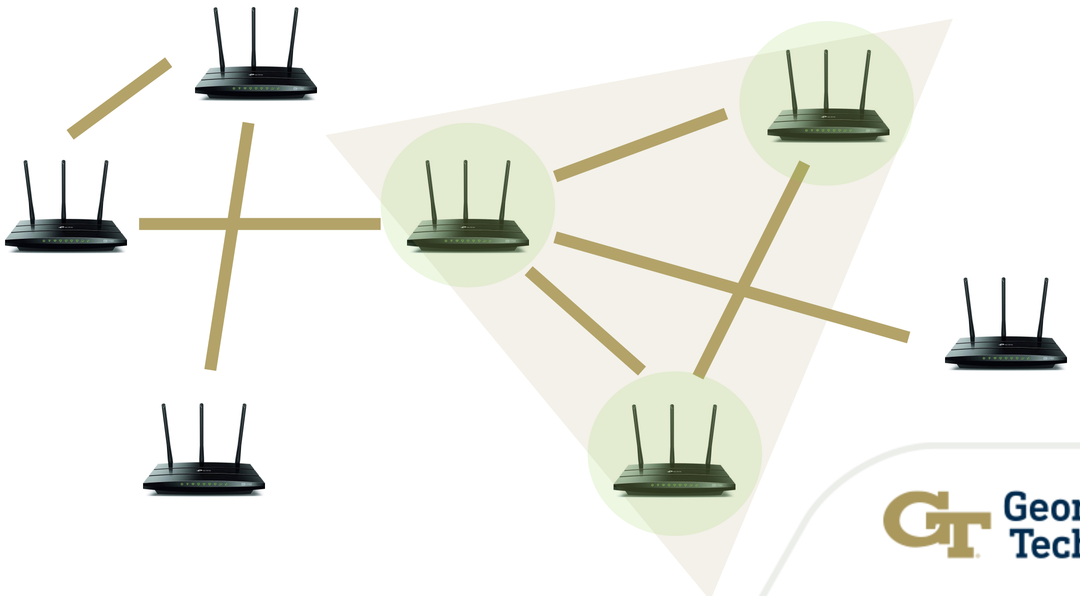
We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

<u>Triangle Centrality</u>:
finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

That concentration of triangles is a rep. of augmented network density, permitting the flow and spread of the influence of data more rapidly through the connections

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Georgia Tech

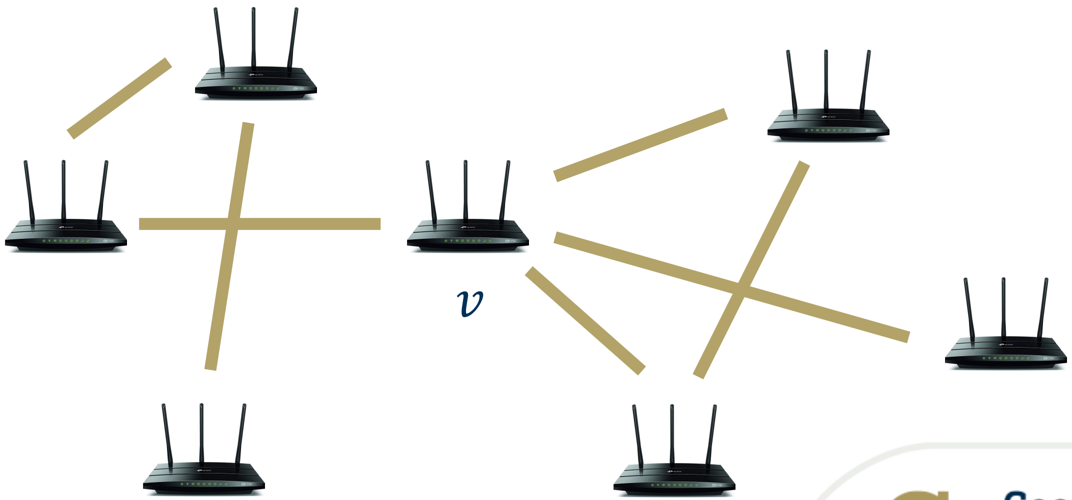
Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with $|V|$ **vertices** $|E|$ **edges**

where

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

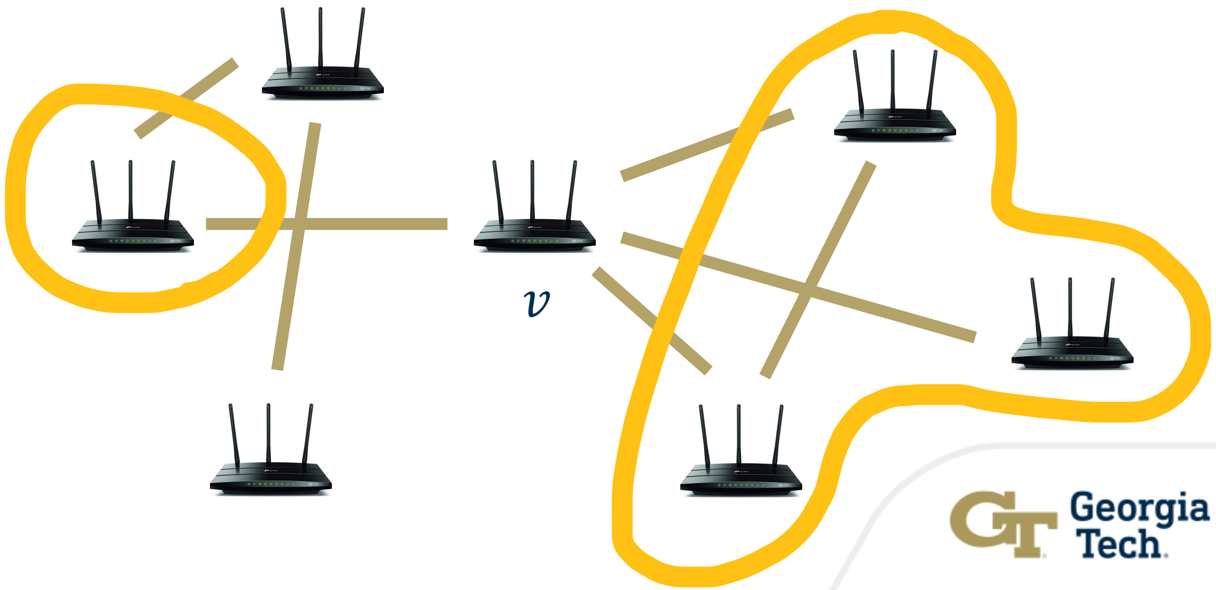
Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with $|V|$ **vertices** $|E|$ **edges**

where

$N(v)$ **is the set of neighbors of** v

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

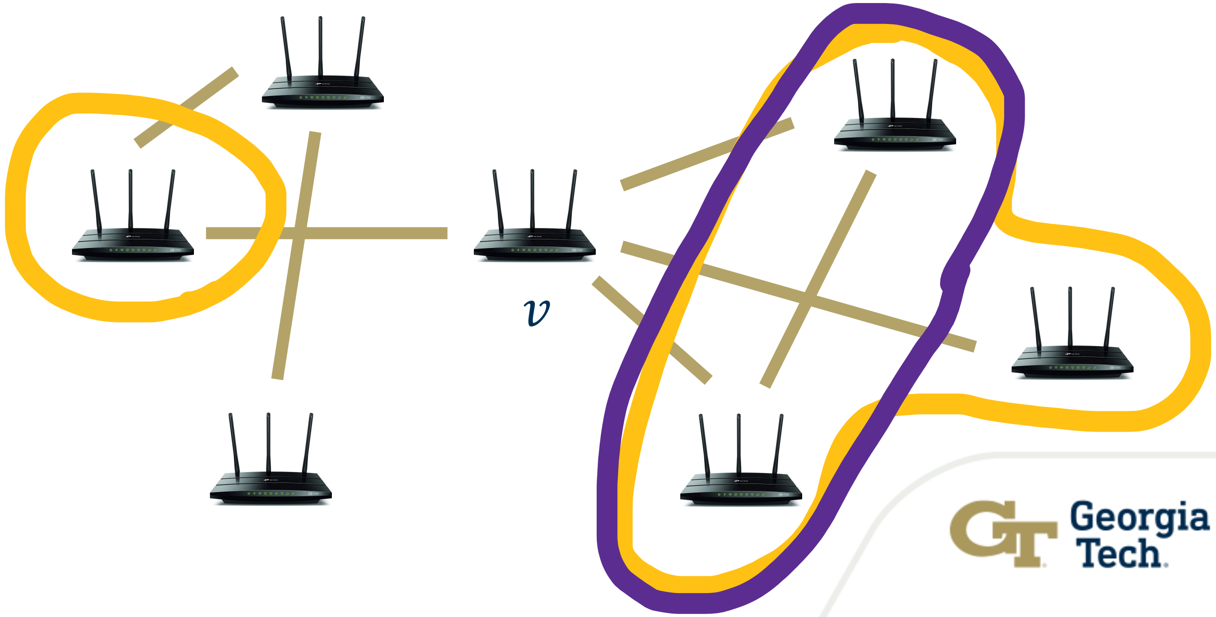
Given an undirected network of entities, a graph $G = (V, E)$ with $|V|$ **vertices** $|E|$ **edges**

where

$N(v)$ **is the set of neighbors of** v

$N_\Delta(v)$ **is the set of neighbors in triangles with** v

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with $|V|$ **vertices** $|E|$ **edges**

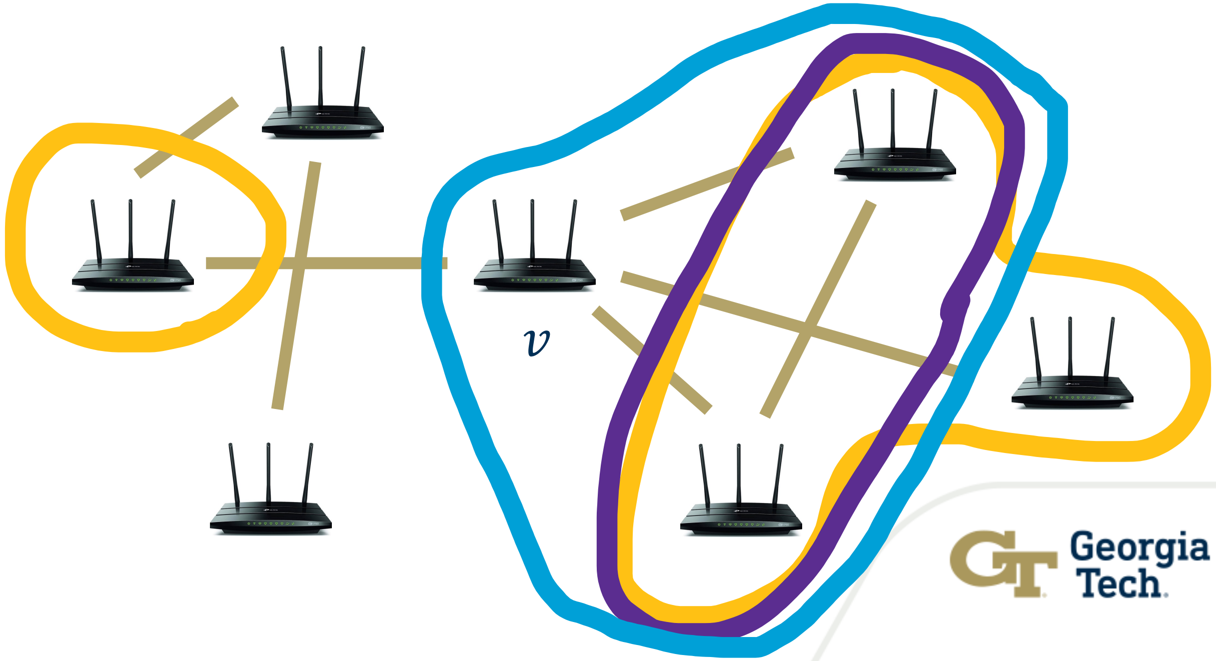
where

$N(v)$ **is the set of neighbors of** v

$N_\Delta(v)$ **is the set of neighbors in triangles with** v

$N_\Delta^+(v)$ **is the closed set including** v

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with $|V|$ **vertices** $|E|$ **edges**

where

$N(v)$ **is the set of neighbors of** v

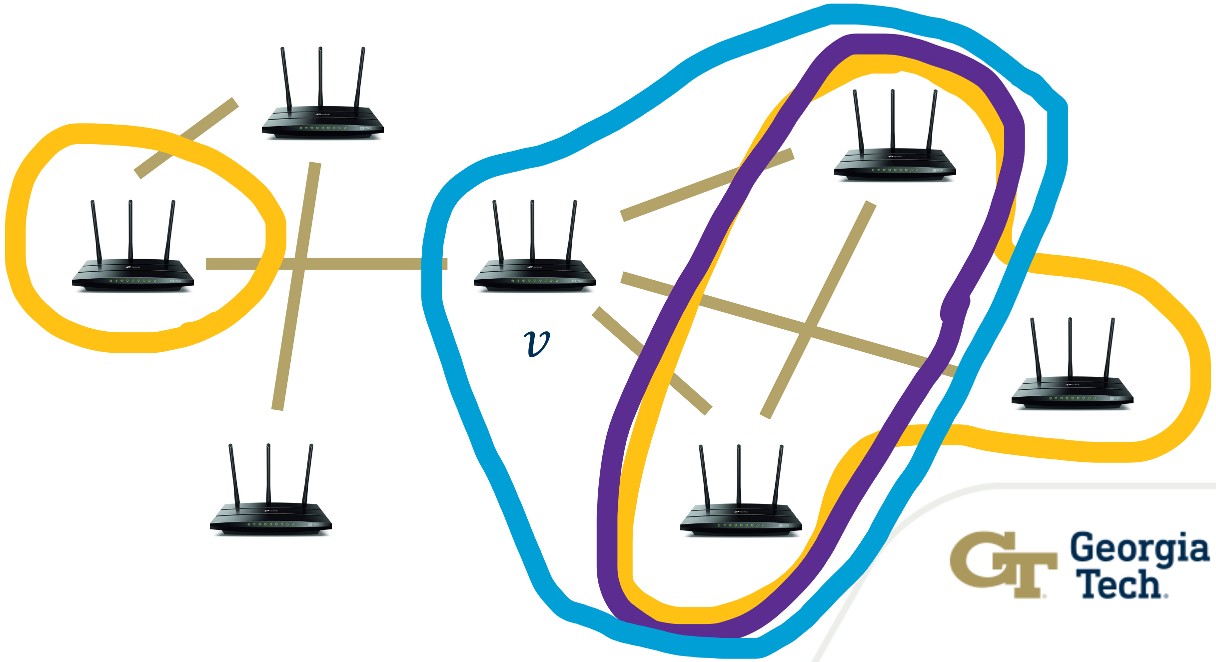
$N_\Delta(v)$ **is the set of neighbors in triangles with** v

$N_\Delta^+(v)$ **is the closed set including** v

$\Delta(v)$ **is the triangle count of** v

$\Delta(G)$ **is the triangle count of** G

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

where

$N(v)$ is the set of neighbors of v

$N_\Delta(v)$ is the set of neighbors in triangles with v

$N_\Delta^+(v)$ is the closed set including v

$\Delta(v)$ is the triangle count of v

$\Delta(G)$ is the triangle count of G

$$TC(v) = \frac{\overbrace{\frac{1}{3}\sum_{u \in N_\Delta^+(v)} \Delta(u)}^{\text{sum of } \Delta \text{ for triangle neighbors}} + \overbrace{\sum_{w \in \{N(v) \setminus N_\Delta(v)\}} \Delta(w)}^{\text{sum of } \Delta \text{ for non-triangle neighbors}}}{\Delta(G)}$$

Georgia Tech

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function $\textsc{TriCentActor}(L)$ ▷ Perform the main computation
  $\Delta(G) = \textsc{TriCountActor}(L)$ ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_\Delta(v)\}$ **then** ▷ Compute $\dfrac{\frac{1}{3}\sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$
▷ Send an active message to process1 (non-blocking)
      $Actor_p.\text{send}(1, \textsc{FindOwner}(L_u), u, v, \Delta(G), \frac{1}{3})$
    else ▷ Compute $\dfrac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(u)}{\Delta(G)}$
      $Actor_p.\text{send}(1, \textsc{FindOwner}(L_u), u, v, \Delta(G), 1)$
    $TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$
  $\textsc{Wait}()$ ▷ Wait for the completion of local send/recv
  return TC
function $\textsc{TriCentActorProcess1}(u, v, \Delta(G), mult)$
▷ The process1 message handler
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$
▷ Send an active message to process2 (non-blocking)
  $Actor_p.\text{send}(2, \textsc{FindOwner}(L_v), v, m_{tc})$
function $\textsc{TriCentActorProcess2}(v, m_{tc})$
▷ The process2 message handler
  $TC(v)$ += m_{tc}

Notation: Let r_p be the local rank, $\hat{L}$ be the local rows of L owned by r_p, c be the local counter owned by r_p, $Actor_p$ denote the actor instance that is running on r_p, N_Δ be the local array of lists holding set of neighbors in triangles with local rows owned by r_p, TC be the local rows of the triangle centrality matrix owned by r_p, $\Delta(G)$ be the global triangle count.

`send` **Semantics:** `Actor.send(proc handler, rank, packet)`

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TriCentActor(L) ▷ Perform the main computation
  $\Delta(G) = $ TriCountActor(L) ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_\Delta(v)\}$ **the**
  ▷ Send an active message to
      $Actor_p$.send(1,F
    else
      $Actor_p$.send(1,F
    $TC(v)$ += $\frac{1}{3}\Delta(v)/$
  Wait() ▷
  return TC
function TriCentActorP
  ▷ The process1 message han
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta$
  ▷ Send an active message to
  $Actor_p$.send(2,FindOw
function TriCentActorP
  ▷ The process2 message han
  $TC(v)$ += m_{tc}

Notation: Let r_p be the local rank, $\hat{L}$ be the local rows of L owned by r_p, c be the local counter owned by r_p, $Actor_p$ denote the actor instance that is running on r_p, N_Δ be the local array of lists holding set of neighbors in triangles with local rows owned by r_p, TC be the local rows of the ... r_p, $\Delta(G)$ be the global triangle count. ...c handler, rank, packet)

```
function TriCountActor(L)
    for {l_vw, l_vu ∈ L̂, u < w < v} do
  ▷ Send an active message to process1 (non-blocking)
        Actor_p.send(1,FindOwner(L_wu), v, w, u)
    Wait()
    return AllReduce(c)
function TriCountActorProcess1(v, w, u)
    if {l_wu ∈ L̂} then
        c += 1
  ▷ Send an active message to process2 (non-blocking)
        Actor_p.send(2,FindOwner(L_v), v, w, u)
        Actor_p.send(2,FindOwner(L_w), w, v, u)
        Actor_p.send(2,FindOwner(L_u), u, v, w)
function TriCountActorProcess2(i, j, k)
    Δ(i) += 1
    N_Δ(i) ← N_Δ(i) ∪ {j, k}
function FindOwner(row)        ▷ Returns a rank responsible for row
    return row % P             ▷ 1-D Cyclic distribution
```

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TriCentActor(L) ▷ Perform the main computation
    $\Delta(G) = $ TriCountActor(L) ▷ Compute global tricount
    for $\{l_{vu} \in \hat{L}\}$ **do**
        if $\{u \in N_\Delta(v)\}$ **the**
▷ Send an active message to
            $Actor_p$.send(1,F
        else
            $Actor_p$.send(1,F
        $TC(v)$ += $\frac{1}{3}\Delta(v)/$
    Wait() ▷
    return TC
function TriCentActorP
▷ The process1 message han
    $m_{tc} \leftarrow mult * \Delta(u)/\Delta$
▷ Send an active message to
    $Actor_p$.send(2,FindOw
function TriCentActorP
▷ The process2 message han
    $TC(v)$ += m_{tc}

function TriCountActor(L)
    for $\{l_{vw}, l_{vu} \in \hat{L}, u < w < v\}$ **do**
▷ Send an active message to process1 (non-blocking)
        $Actor_p$.send(1,FindOwner(L_{wu}), v, w, u)
    Wait()
    return AllReduce(c)
function TriCountActorProcess1(v, w, u)
    if $\{l_{wu} \in \hat{L}\}$ **then**
        c += 1
▷ Send an active message to process2 (non-blocking)
        $Actor_p$.send(2,FindOwner(L_v), v, w, u)
        $Actor_p$.send(2,FindOwner(L_w), w, v, u)
        $Actor_p$.send(2,FindOwner(L_u), u, v, w)
function TriCountActorProcess2(i, j, k)
    $\Delta(i)$ += 1
    $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$
function FindOwner(row) ▷ Returns a rank responsible for row
    return row % P ▷ 1-D Cyclic distribution

Notation
be th
is ru
r_p, $\Delta(G)$ be the global triangle count.
c handler, rank, packet)

Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle

Georgia Tech

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
$\Delta(G)$ = TRICOUNTACTOR(L) ▷ Compute global tricount
for $\{l_{vu} \in \hat{L}\}$ **do**
if $\{u \in N_\Delta(v)\}$ **the**
▷ Send an active message to
$Actor_p$.send(1,F
else
$Actor_p$.send(1,F
$TC(v)$ += $\frac{1}{3}\Delta(v)/$
WAIT()
return TC
function TRICENTACTORP
▷ The process1 message han
$m_{tc} \leftarrow mult * \Delta(u)/\Delta$
▷ Send an active message to
$Actor_p$.send(2,FINDOW
function TRICENTACTORP
▷ The process2 message han
$TC(v)$ += m_{tc}

Notation

function TRICOUNTACTOR(L)
for $\{l_{vw}, l_{vu} \in \hat{L}, u < w < v\}$ **do**
▷ Send an active message to process1 (non-blocking)
$Actor_p$.send(1,FINDOWNER(L_{wu}), v, w, u)
WAIT()
return ALLREDUCE(c)
function TRICOUNTACTORPROCESS1(v, w, u)
if $\{l_{wu} \in \hat{L}\}$ **then**
c += 1
▷ Send an active message to process2 (non-blocking)
$Actor_p$.send(2,FINDOWNER(L_v), v, w, u)
$Actor_p$.send(2,FINDOWNER(L_w), w, v, u)
$Actor_p$.send(2,FINDOWNER(L_u), u, v, w)
function TRICOUNTACTORPROCESS2(i, j, k)
$\Delta(i)$ += 1
$N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$
function FINDOWNER(row) ▷ Returns a rank responsible for row
return row % P ▷ 1-D Cyclic distribution

Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle

Send a non-blocking fine-grained active message to remote vertex owner

MSG HANDLER

Georgia Tech

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
  $\Delta(G)$ = TRICOUNTACTOR(L) ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_\Delta(v)\}$ **the**
  ▷ Send an active message to
      $Actor_p$.send(1,F
    else
      $Actor_p$.send(1,F
    $TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta$
  WAIT() ▷
  return TC
function TRICENTACTORP
  ▷ The process1 message han
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta$
  ▷ Send an active message to
  $Actor_p$.send(2,FINDOW
function TRICENTACTORP
  ▷ The process2 message han
  $TC(v)$ += m_{tc}

Notation
be t
is u

function TRICOUNTACTOR(L)
  for $\{l_{vw}, l_{vu} \in \hat{L}, u < w < v\}$ **do**
  ▷ Send an active message to process1 (non-blocking)
    $Actor_p$.send(1,FINDOWNER(L_{wu}), v, w, u)
  WAIT()
  return ALLREDUCE(c)
function TRICOUNTACTORPROCESS1(v, w, u)
  if $\{l_{wu} \in \hat{L}\}$ **then**
    c += 1
  ▷ Send an active message to process2 (non-blocking)
    $Actor_p$.send(2,FINDOWNER(L_v), v, w, u)
    $Actor_p$.send(2,FINDOWNER(L_w), w, v, u)
    $Actor_p$.send(2,FINDOWNER(L_u), u, v, w)
function TRICOUNTACTORPROCESS2(i, j, k)
  $\Delta(i)$ += 1
  $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$
function FINDOWNER(row) ▷ Returns a rank responsible for row
  return row % P ▷ 1-D Cyclic distribution

Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle

Send a non-blocking fine-grained active message to remote vertex owner

MSG HANDLER

If triangle present, increment local counter and send non-blocking fine-grained active messages to each of v, w, u to update their vertex triangle count and triangle neighbor lists

MSG HANDLER

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TriCentActor(L) ▷ Perform the main computation
 $\Delta(G)$ = TriCountActor(L) ▷ Compute global tricount
 for $\{l_{vu} \in \hat{L}\}$ **do**
 if $\{u \in N_\Delta(v)\}$ **the**
 ▷ Send an active message to
 $Actor_p$.send(1,F
 else
 $Actor_p$.send(1,F
 $TC(v)$ += $\frac{1}{3}\Delta(v)/$
 ▷ Send an active message to
 $Actor_p$.send(2,FindOw
function TriCentActorP
 ▷ The process2 message han
 $TC(v)$ += m_{tc}

Notation

function TriCountActor(L)
 for $\{l_{vw}, l_{vu} \in \hat{L}, u < w < v\}$ **do**
 ▷ Send an active message to process1 (non-blocking)
 $Actor_p$.send(1,FindOwner(L_{wu}), v, w, u)
 Wait()
 return AllReduce(c)
function TriCountActorProcess1(v, w, u)
 if $\{l_{wu} \in \hat{L}\}$ **then**
 c += 1
 ▷ Send an active message to process2 (non-blocking)
 $Actor_p$.send(2,FindOwner(L_v), v, w, u)
 $Actor_p$.send(2,FindOwner(L_w), w, v, u)
 $Actor_p$.send(2,FindOwner(L_u), u, v, w)
function TriCountActorProcess2(i, j, k)
 $\Delta(i)$ += 1
 $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$
function FindOwner(row) ▷ Returns a rank responsible for row
 return row % P ▷ 1-D Cyclic distribution

Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle

Send a non-blocking fine-grained active message to remote vertex owner

MSG HANDLER

Await completion of all local sends and receives

If triangle present, increment local counter and send non-blocking fine-grained active messages to each of v, w, u to update their vertex triangle count and triangle neighbor lists

MSG HANDLER

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
$\Delta(G) = $ TRICOUNTACTOR(L) ▷ Compute global tricount
for $\{l_{vu} \in \hat{L}\}$ **do**
if $\{u \in N_\Delta(v)\}$ **the**
▷ Send an active message to
$Actor_p$.send(1,F
else
$Actor_p$.send(1,F
$TC(v)$ += $\frac{1}{3}\Delta(v)/$
▷ Send an active message to
$Actor_p$.send(2,FINDOW

Notation
be th
is lu

```
function TRICOUNTACTOR(L)
    for {l_vw, l_vu ∈ L̂, u < w < v} do
        ▷ Send an active message to process1 (non-blocking)
        Actor_p.send(1,FINDOWNER(L_wu), v, w, u)
    WAIT()
    return ALLREDUCE(c)
function TRICOUNTACTORPROCESS1(v, w, u)
    if {l_wu ∈ L̂} then
        c += 1
        ▷ Send an active message to process2 (non-blocking)
        Actor_p.send(2,FINDOWNER(L_v), v, w, u)
        Actor_p.send(2,FINDOWNER(L_w), w, v, u)
        Actor_p.send(2,FINDOWNER(L_u), u, v, w)
function TRICOUNTACTORPROCESS2(i, j, k)
    Δ(i) += 1
    N_Δ(i) ← N_Δ(i) ∪ {j, k}
function FINDOWNER(row)          ▷ Returns a rank responsible for row
    return row % P               ▷ 1-D Cyclic distribution
```

Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle

Send a non-blocking fine-grained active message to remote vertex owner

MSG HANDLER

If triangle present, increment local counter and send non-blocking fine-grained active messages to each of v, w, u to update their vertex triangle count and triangle neighbor lists

MSG HANDLER

Await completion of all local sends and receives

Returns global reduced triangle count

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
  $\Delta(G) = $ TRICOUNTACTOR(L) ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_{\Delta}(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N_{\Delta}^{+}(v)} \Delta(u)}{\Delta(G)}$
      ▷ Send an active message to process1 (non-blocking)
      $Actor_p$.send(1,FINDOWNER(L_u)$, u, v, \Delta(G), \frac{1}{3}$)
    else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_{\Delta}(v)\}} \Delta(u)}{\Delta(G)}$
      $Actor_p$.send(1,FINDOWNER(L_u)$, u, v, \Delta(G), 1$)
    $TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$
  WAIT() ▷ Wait for the completion of local send/recv
  return TC

function TRICENTACTORPROCESS1($u, v, \Delta(G), mult$)
  ▷ The process1 message handler
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$
  ▷ Send an active message to process2 (non-blocking)
  $Actor_p$.send(2,FINDOWNER(L_v)$, v, m_{tc}$)

function TRICENTACTORPROCESS2(v, m_{tc})
  ▷ The process2 message handler
  $TC(v)$ += m_{tc}

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
  $\Delta(G)$ = TRICOUNTACTOR(L) ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_\Delta(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N^+_\Delta(v)} \Delta(u)}{\Delta(G)}$
  ▷ Send an active message to process1 (non-blocking)
      $Actor_p$.send(1,FINDOWNER(L_u), $u, v, \Delta(G), \frac{1}{3}$)
    else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(u)}{\Delta(G)}$
      $Actor_p$.send(1,FINDOWNER(L_u), $u, v, \Delta(G), 1$)
    $TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$
  WAIT() ▷ Wait for the completion of local send/recv
  return TC
function TRICENTACTORPROCESS1($u, v, \Delta(G), mult$)
  ▷ The process1 message handler
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$
  ▷ Send an active message to process2 (non-blocking)
  $Actor_p$.send(2,FINDOWNER(L_v), v, m_{tc})
function TRICENTACTORPROCESS2(v, m_{tc})
  ▷ The process2 message handler
  $TC(v)$ += m_{tc}

Sum of the Δ for the triangle neighbors
Send non-blocking message to remote vertex owner

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
  $\Delta(G) =$ TRICOUNTACTOR(L) ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_\Delta(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$
  ▷ Send an active message to process1 (non-blocking)
      $Actor_p$.send(1,FINDOWNER(L_u)$, u, v, \Delta(G), \frac{1}{3}$)
    else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(v)}{\Delta(G)}$
      $Actor_p$.send(1,FINDOWNER(L_u)$, u, v, \Delta(G), 1$)
    $TC(v) \mathrel{+}= \frac{1}{3}\Delta(v)/\Delta(G)$
  WAIT() ▷ Wait for the completion of local ser... v
  return TC
function TRICENTACTORPROCESS1($u, v, \Delta(G), mult$)
  ▷ The process1 message handler
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$
  ▷ Send an active message to process2 (non-blocking)
  $Actor_p$.send(2,FINDOWNER(L_v)$, v, m_{tc}$)
function TRICENTACTORPROCESS2(v, m_{tc})
  ▷ The process2 message handler
  $TC(v) \mathrel{+}= m_{tc}$

Sum of the Δ for the triangle neighbors
Send non-blocking message to remote vertex owner

Sum of the Δ for the non-triangle neighbors
Send non-blocking message to remote vertex owner

MSG HANDLER

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation
  $\Delta(G) =$ TRICOUNTACTOR(L) ▷ Compute global tricount
  for $\{l_{vu} \in \hat{L}\}$ **do**
    if $\{u \in N_\Delta(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$
  ▷ Send an active message to process1 (non-blocking)
      $Actor_p$.send(1,FINDOWNER(L_u)$, u, v, \Delta(G), \frac{1}{3}$)
    else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(v)}{\Delta(G)}$
      $Actor_p$.send(1,FINDOWNER(L_u)$, u, v, \Delta(G), 1$)
    $TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$
  WAIT() ▷ Wait for the completion of local ser...v
  return TC
function TRICENTACTORPROCESS1($u, v, \Delta(G), mult$)
  ▷ The process1 message handler
  $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$
  ▷ Send an active message to process2 (non-blocking)
  $Actor_p$.send(2,FINDOWNER(L_v)$, v, m_{tc}$)
function TRICENTACTORPROCESS2(v, m_{tc})
  ▷ The process2 message handler
  $TC(v)$ += m_{tc}

Sum of the Δ for the triangle neighbors
Send non-blocking message to remote vertex owner

Sum of the Δ for the non-triangle neighbors
Send non-blocking message to remote vertex owner

Addition of the vertex within the final triangle centrality

MSG HANDLER

Georgia Tech

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation

$\Delta(G)$ = TRICOUNTACTOR(L) ▷ Compute global tricount

for $\{l_{vu} \in \hat{L}\}$ **do**

if $\{u \in N_{\Delta}(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N_{\Delta}^{+}(v)} \Delta(u)}{\Delta(G)}$

▷ Send an active message to process1 (non-blocking)

$Actor_p$.send(1,FINDOWNER(L_u), $u, v, \Delta(G), \frac{1}{3}$)

else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_{\Delta}(v)\}} \Delta(v)}{\Delta(G)}$

$Actor_p$.send(1,FINDOWNER(L_u), $u, v, \Delta(G), 1$)

$TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$

WAIT() ▷ Wait for the completion of local ser…v

return TC

function TRICENTACTORPROCESS1($u, v, \Delta(G), mult$)

▷ The process1 message handler

$m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$

▷ Send an active message to process2 (non-blocking)

$Actor_p$.send(2,FINDOWNER(L_v), v, m_{tc})

function TRICENTACTORPROCESS2(v, m_{tc})

▷ The process2 message handler

$TC(v)$ += m_{tc}

Sum of the Δ for the triangle neighbors

Send non-blocking message to remote vertex owner

Sum of the Δ for the non-triangle neighbors

Send non-blocking message to remote vertex owner

Addition of the vertex within the final triangle centrality

MSG HANDLER

Do the calculation depending on packet variables and send a non-blocking message to v to increment final triangle centrality

MSG HANDLER

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TriCentActor(L) ▷ Perform the main computation
 $\Delta(G) =$ TriCountActor(L) ▷ Compute global tricount
 for $\{l_{vu} \in \hat{L}\}$ **do**
  if $\{u \in N_\Delta(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N^+_\Delta(v)} \Delta(u)}{\Delta(G)}$
 ▷ Send an active message to process1 (non-blocking)
   $Actor_p$.send(1,FindOwner(L_u), $u, v, \Delta(G), \frac{1}{3}$)
  else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(v)}{\Delta(G)}$
   $Actor_p$.send(1,FindOwner(L_u), $u, v, \Delta(G), 1$)
  $TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$
 Wait() ▷ Wait for the completion of local ser... ...v
 return TC
function TriCentActorProcess1($u, v, \Delta(G), mult$)
 ▷ The process1 message handler
 ...)
 ...ocess2 (non-blocking)
 ...R(L_v), v, m_{tc})
 ...CESS2(v, m_{tc})
 ▷ The process2 message handler
 $TC(v)$ += m_{tc}

Sum of the Δ for the triangle neighbors
Send non-blocking message to remote vertex owner

Sum of the Δ for the non-triangle neighbors
Send non-blocking message to remote vertex owner

Addition of the vertex within the final triangle centrality

MSG HANDLER

Await completion of all local sends and receives

Do the calculation depending on packet variables and send a non-blocking message to v to increment final triangle centrality

MSG HANDLER

Georgia Tech

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

function TRICENTACTOR(L) ▷ Perform the main computation

$\Delta(G)$ = TRICOUNTACTOR(L) ▷ Compute global tricount

for $\{l_{vu} \in \hat{L}\}$ **do**

if $\{u \in N_\Delta(v)\}$ **then** ▷ Compute $\frac{\frac{1}{3}\sum_{u \in N^+_\Delta(v)} \Delta(u)}{\Delta(G)}$

▷ Send an active message to process1 (non-blocking)

$Actor_p$.send(1,FINDOWNER(L_u), $u, v, \Delta(G), \frac{1}{3}$)

else ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(v)}{\Delta(G)}$

$Actor_p$.send(1,FINDOWNER(L_u), $u, v, \Delta(G), 1$)

$TC(v)$ += $\frac{1}{3}\Delta(v)/\Delta(G)$

WAIT() ▷ Wait for the completion of local se…

return TC

function TRICENT…($u, v, \Delta(G), m_{tc}$)

▷ The process1 message handler

…)

…ocess2 (non-blocking)

…R(L_v), v, m_{tc})

…CESS2(v, m_{tc})

▷ The process2 message handler

$TC(v)$ += m_{tc}

Sum of the Δ for the triangle neighbors

Send non-blocking message to remote vertex owner

Sum of the Δ for the non-triangle neighbors

Send non-blocking message to remote vertex owner

Addition of the vertex within the final triangle centrality

Return triangle centrality

MSG HANDLER

Do the calculation depending on packet variables and send a non-blocking message to v to increment final triangle centrality

MSG HANDLER

Await completion of all local sends and receives

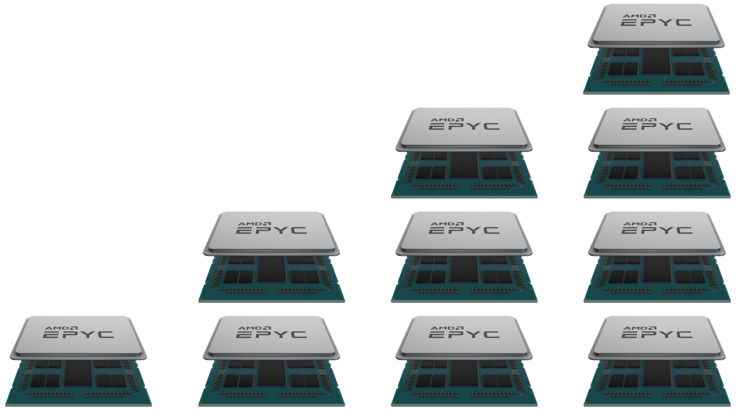
Georgia Tech

Experimental Setup and Architecture

- Experiments conducted on the CPU nodes of
  - The **Perlmutter Supercomputer** at the National Energy Research Scientific Computing Center (NERSC)
    - Each CPU node: 2x AMD EPYC 7763 (Milan) CPUs, 64 physical cores per CPU, 512 GB memory
  - The **HPC PACE Cluster** at Georgia Tech
    - Each CPU node: Dual Intel Xeon Gold 6226 CPUs, 24 total physical cores, 192 GB memory
- We present results using speedup of execution time compared to single core
- We use four large-scale internet network datasets:
  - A **Routers Network Dataset** $(|V| = 191K, |E| = 608K)$
  - An **IP Addresses Network Dataset** $(|V| = 2.3M, |E| = 21.6M)$
  - An **Autonomous Systems Network Dataset** $(|V| = 1.7M, |E| = 11.1M)$
  - A **Synthetic Network Dataset** $($100K rows/core for weak scaling, $|V| = 205M$, $|E| = 13.9B$ at largest scale)
- Results for **different dimensions of scalability** are presented

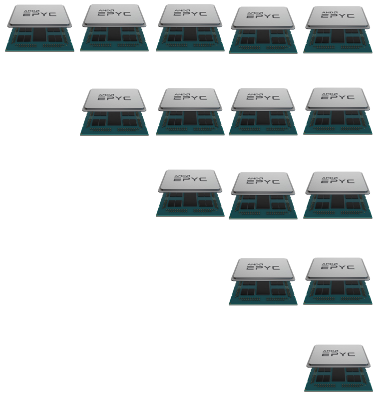
Dimensions of Scalability

(1) SCALE-OUT



increasing #nodes (<u>SCALE-OUT</u>, from 1 to 2,048 cores)

(2) SCALE-UP

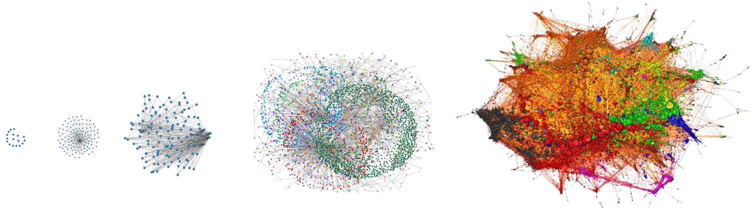


increasing #cores
per node
(<u>SCALE-UP</u>,
from 1
core/node to
128 cores/node)

HARDWARE SCALABILITY

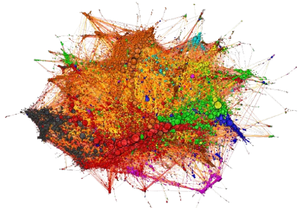
DATASET SCALABILITY

(1) WEAK SCALING



increasing graph size (synthetic dataset, 100K vertices per core)

(2) STRONG SCALING

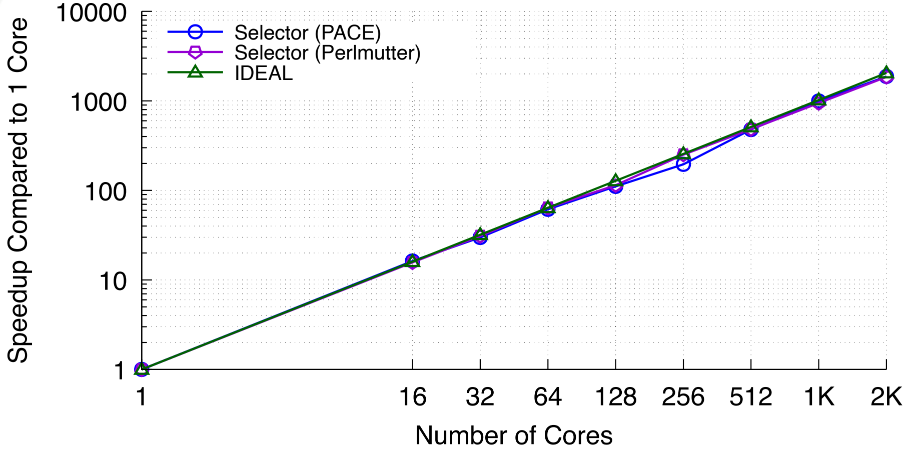


constant graph size
(Routers dataset, IP dataset, ASes dataset)

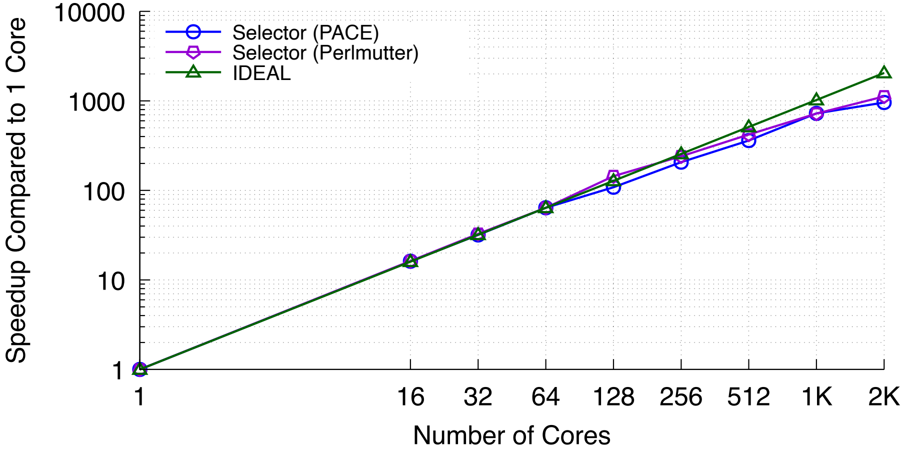
Georgia Tech

Performance Results: Scalability for SCALE-OUT

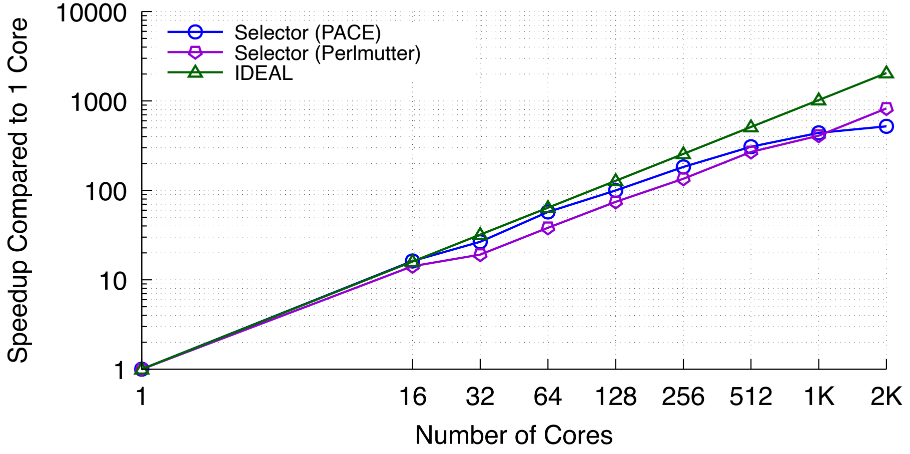
Routers Dataset
Strong Scaling



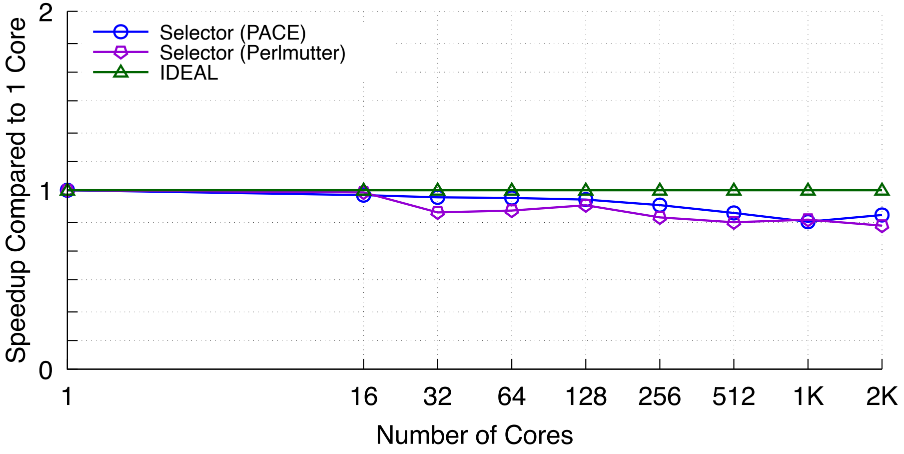
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



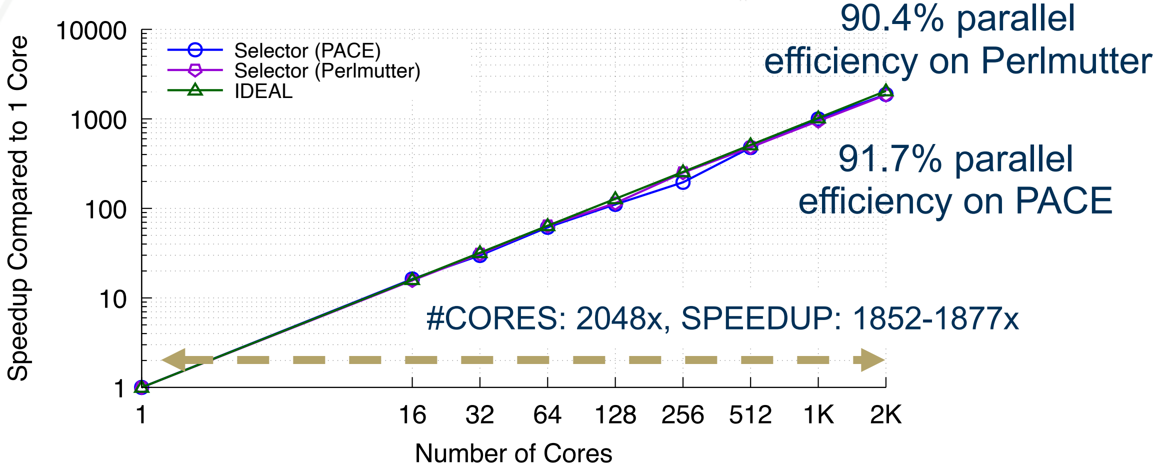
Synthetic Dataset
Weak Scaling



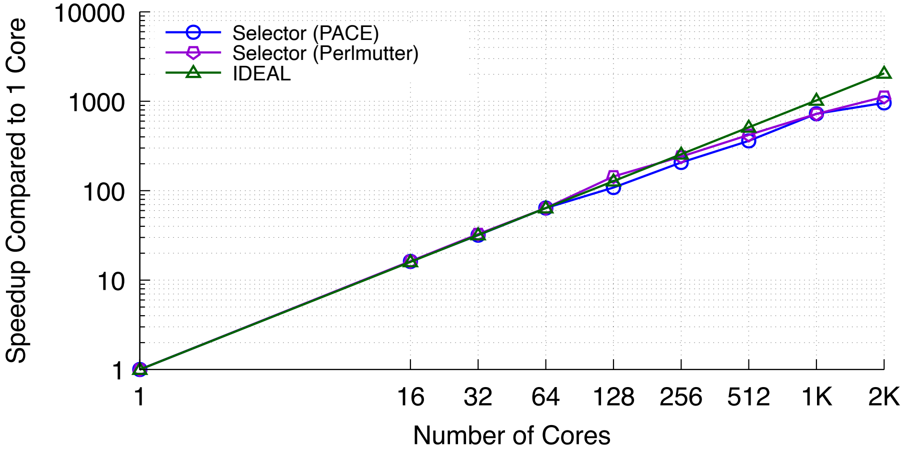
Georgia Tech

Performance Results: Scalability for SCALE-OUT

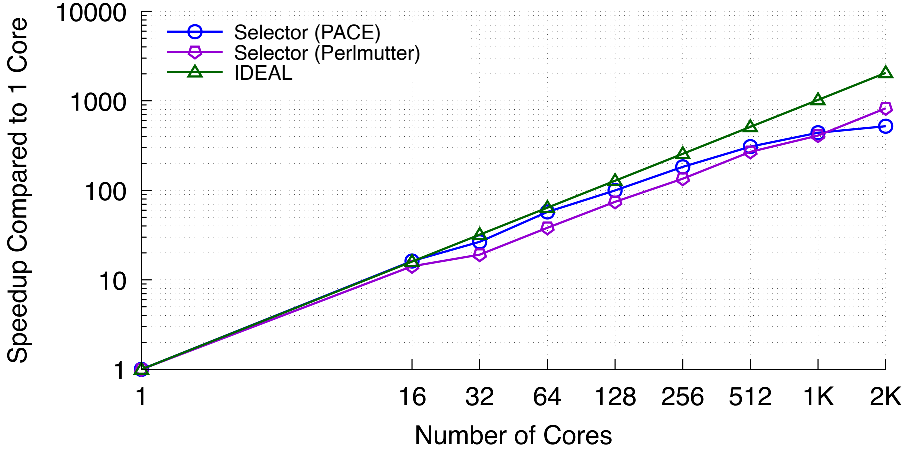
Routers Dataset
Strong Scaling



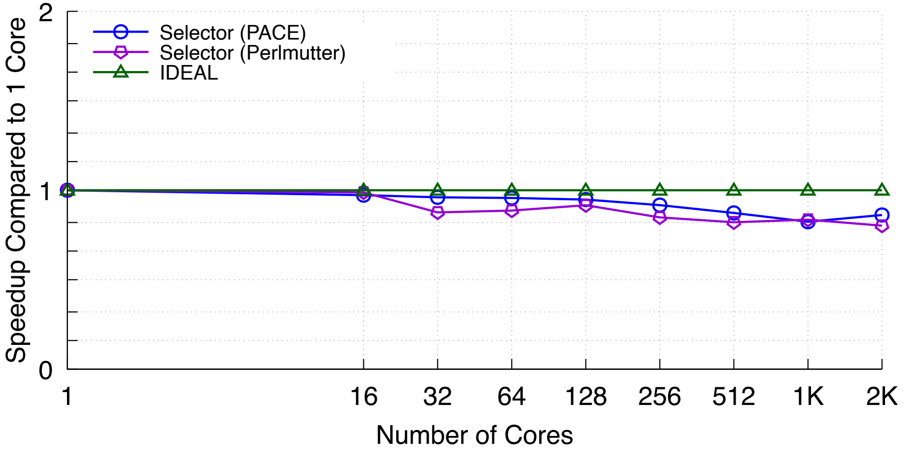
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling

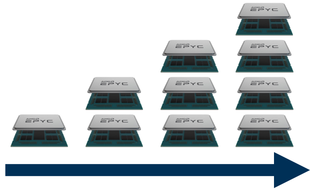


Synthetic Dataset
Weak Scaling

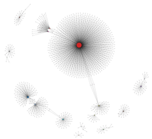


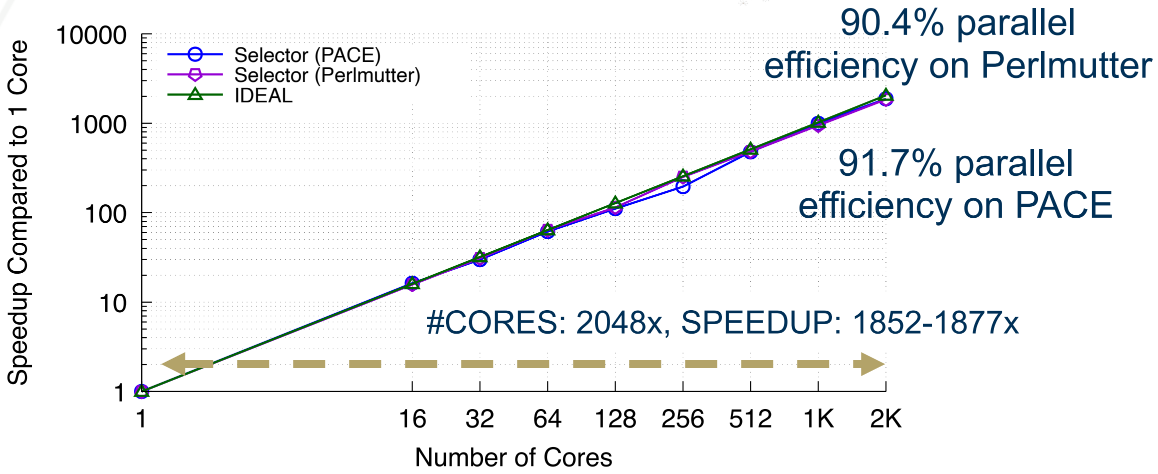
Georgia Tech

Performance Results: Scalability for SCALE-OUT

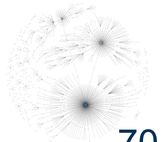


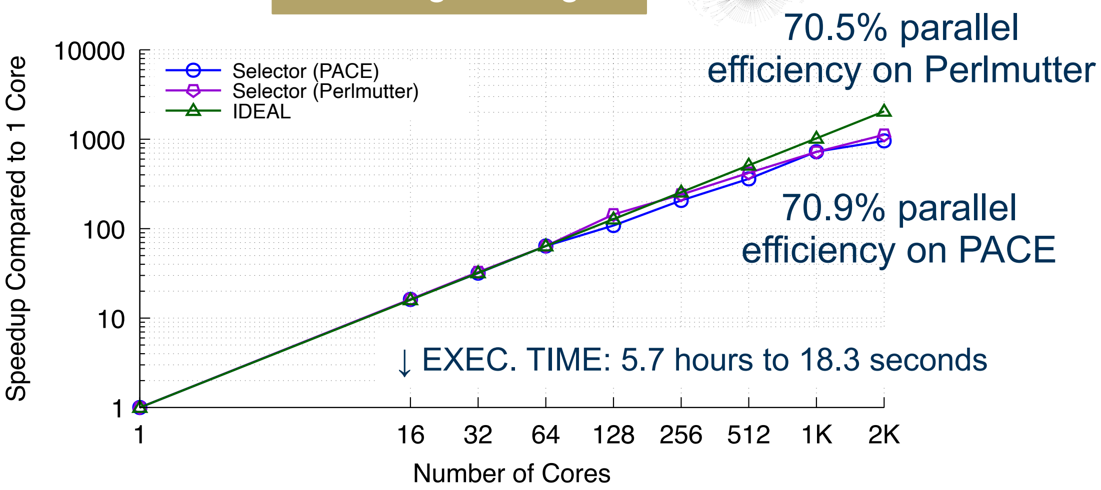
Routers Dataset
Strong Scaling



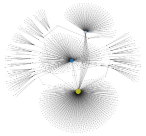


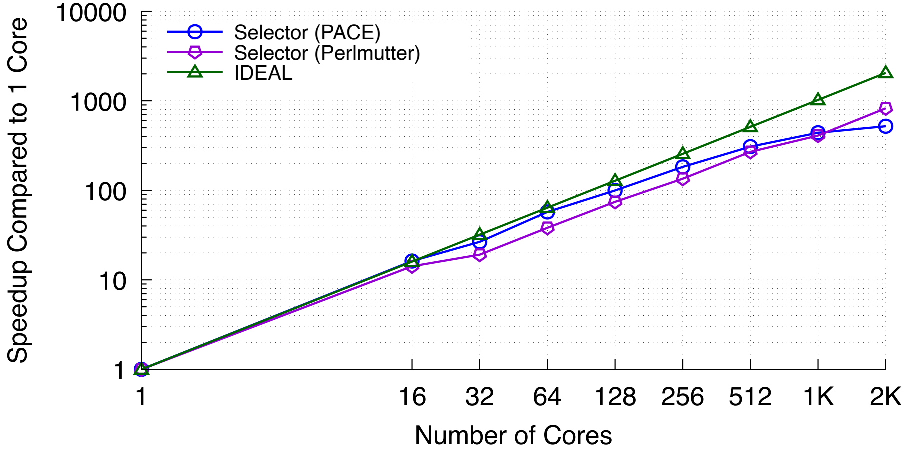
IPs Dataset
Strong Scaling



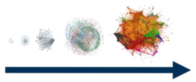


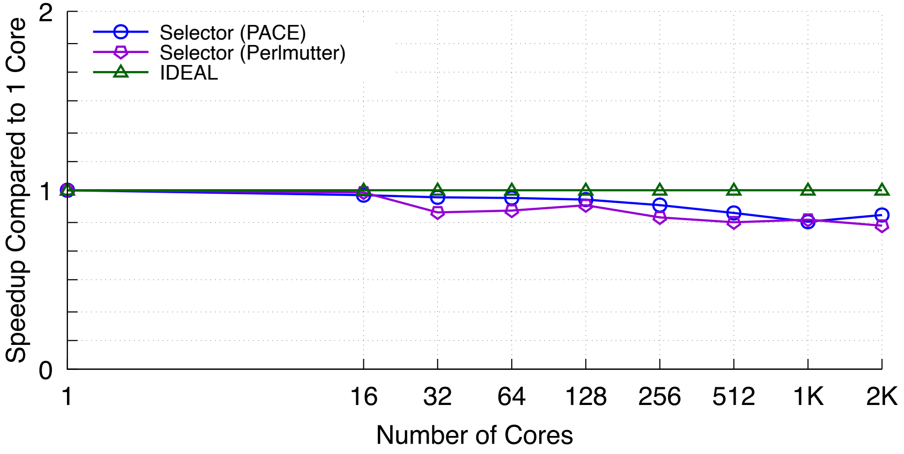
ASes Dataset
Strong Scaling





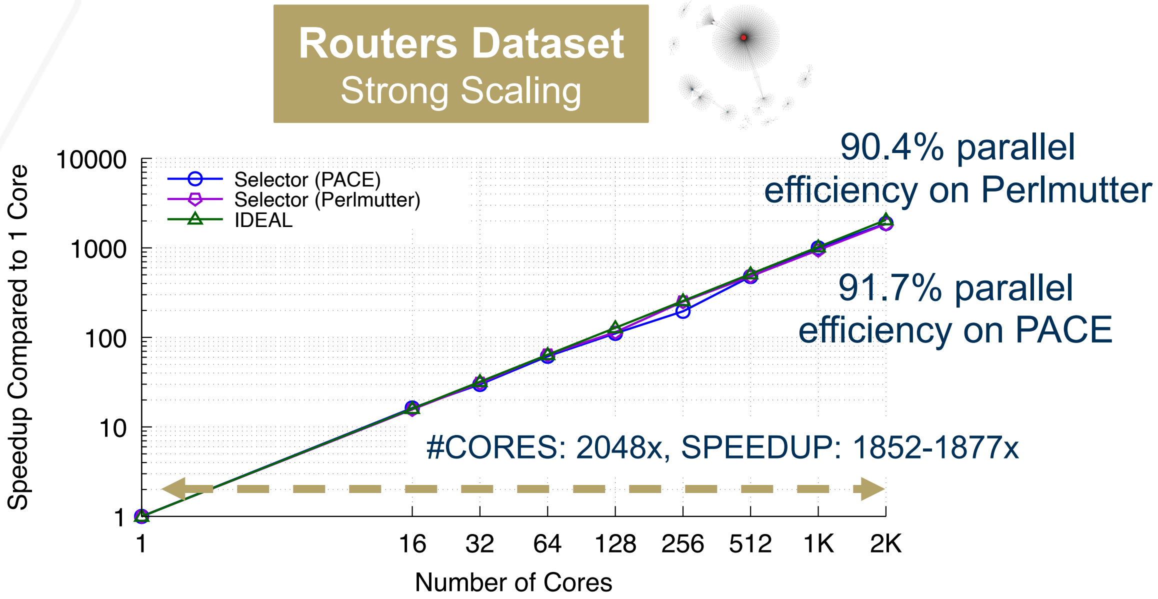
Synthetic Dataset
Weak Scaling

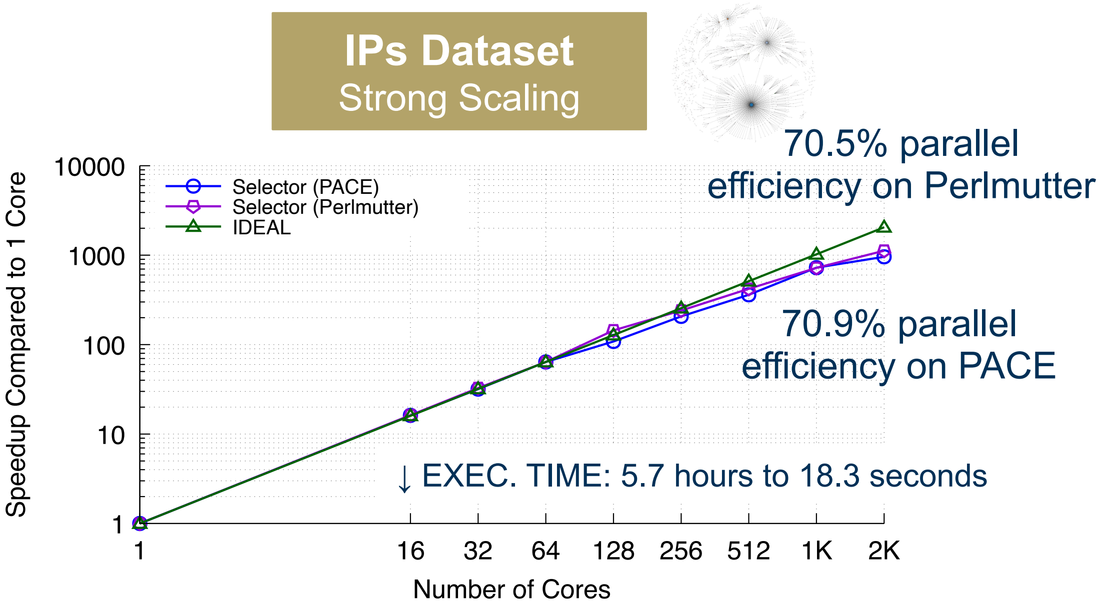


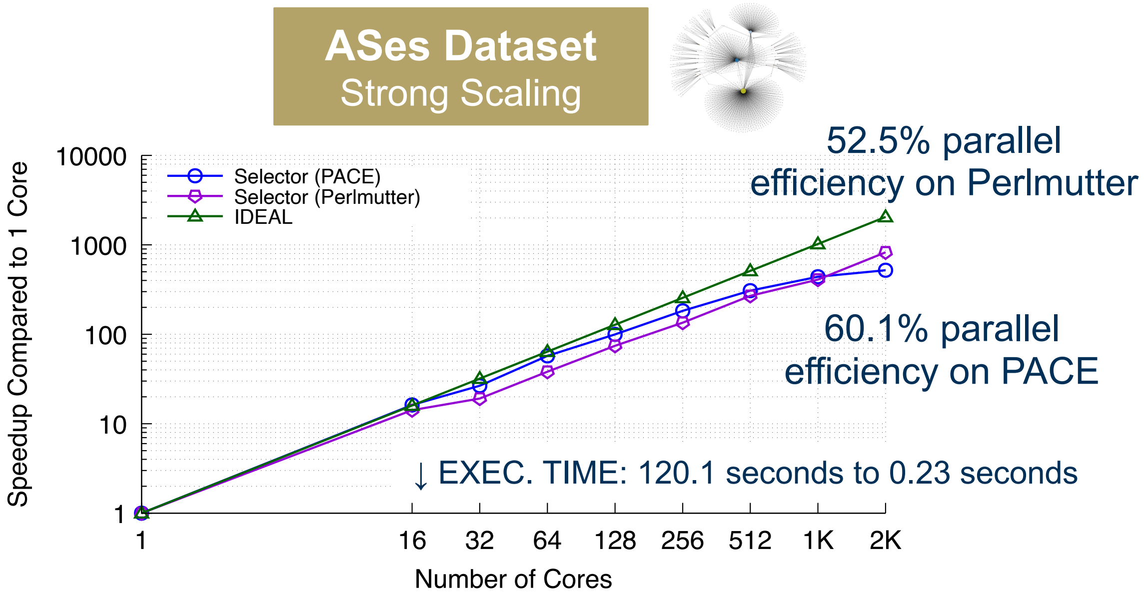


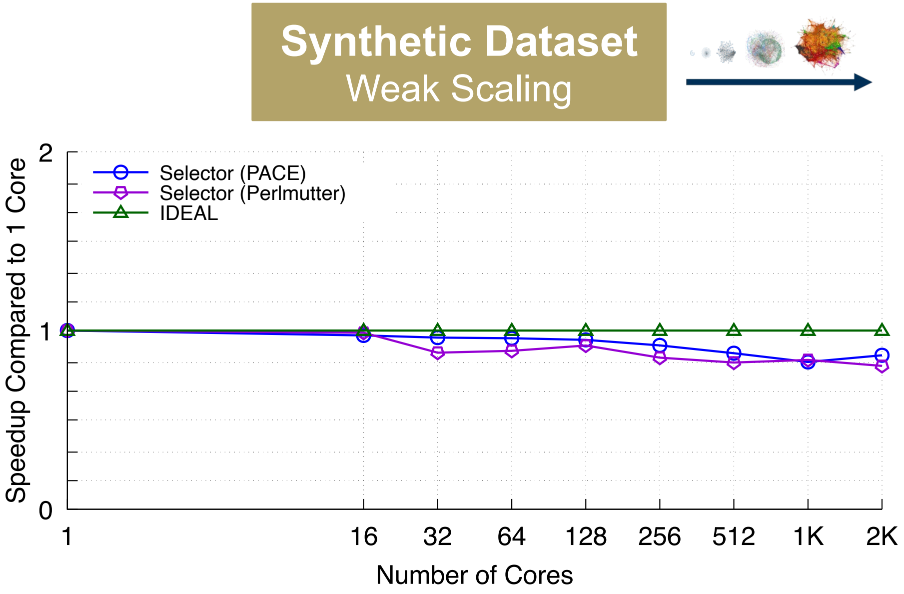
Georgia Tech

Performance Results: Scalability for SCALE-OUT



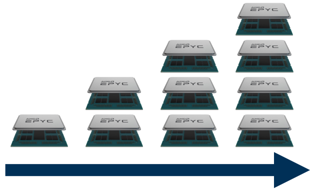




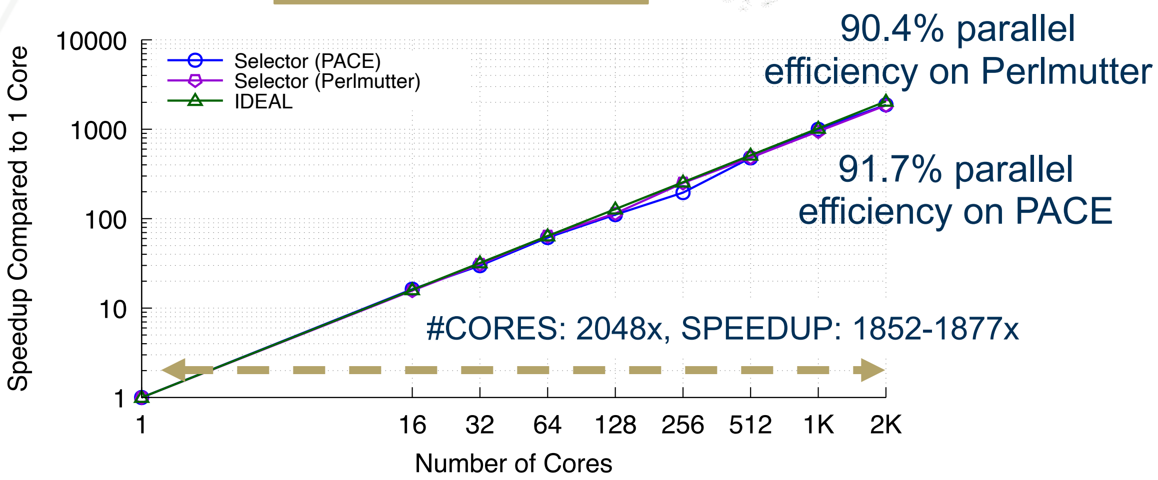


Georgia Tech

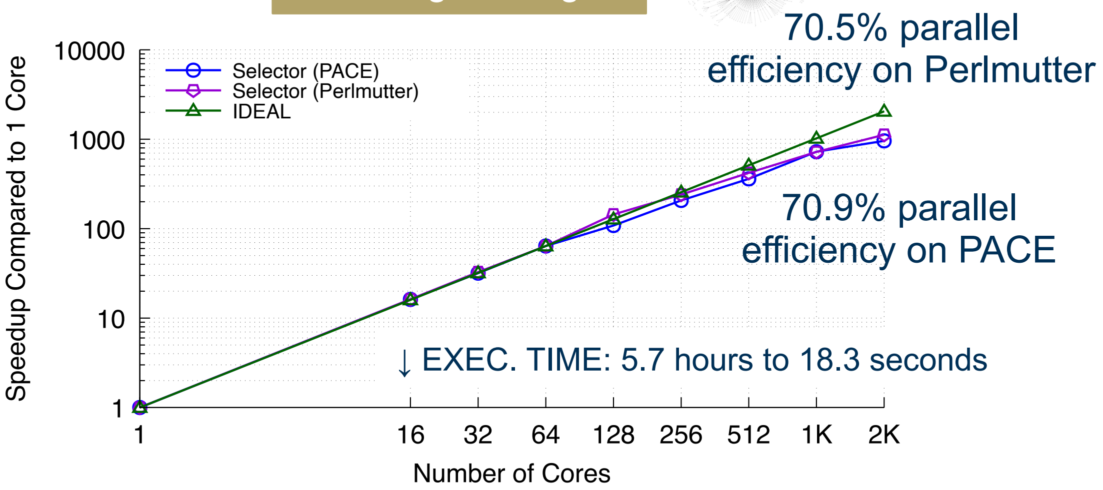
Performance Results: Scalability for SCALE-OUT



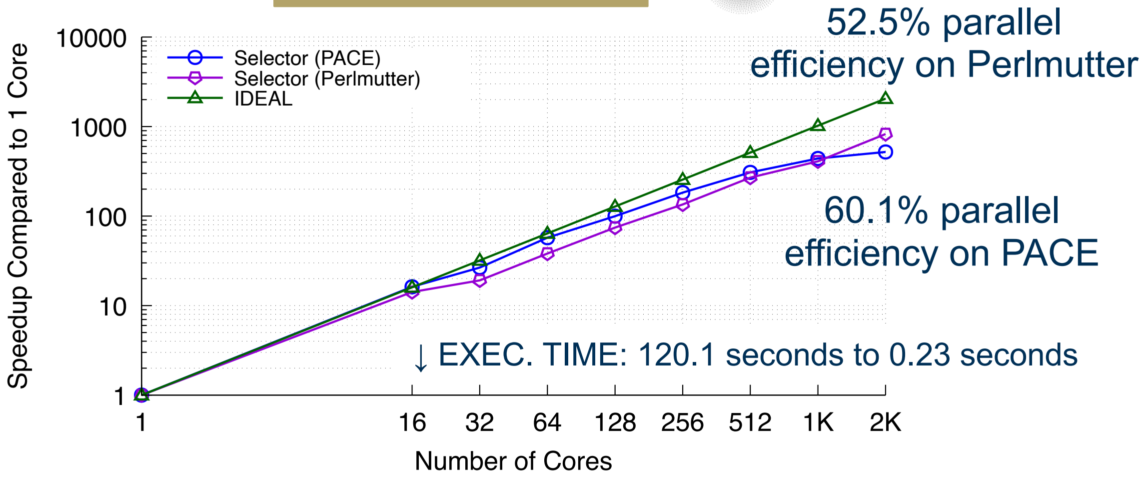
Routers Dataset
Strong Scaling



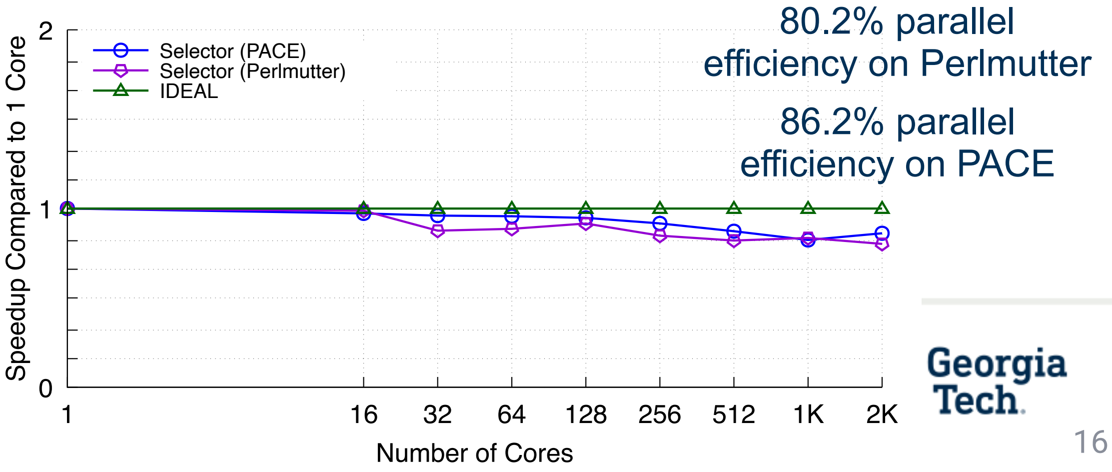
IPs Dataset
Strong Scaling



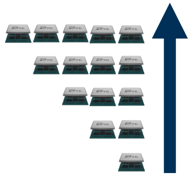
ASes Dataset
Strong Scaling



Synthetic Dataset
Weak Scaling

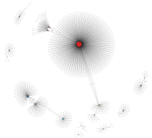


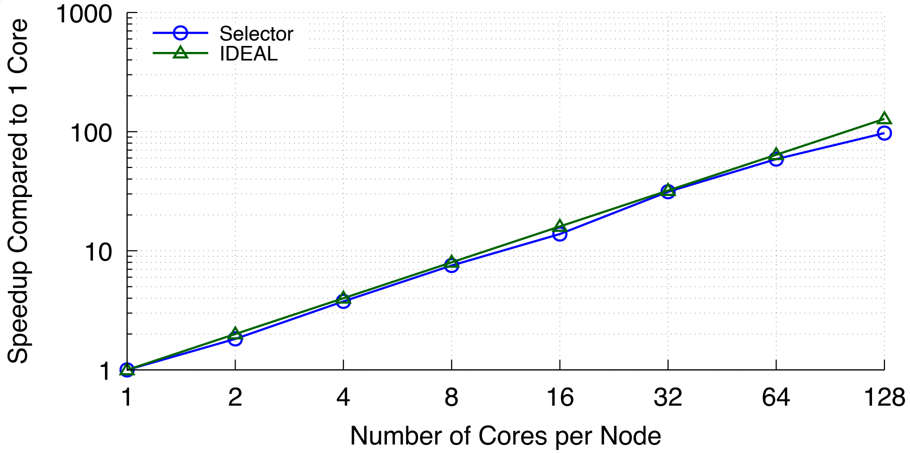
Georgia Tech



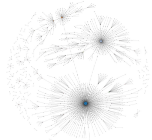
Performance Results: Scalability for SCALE-UP

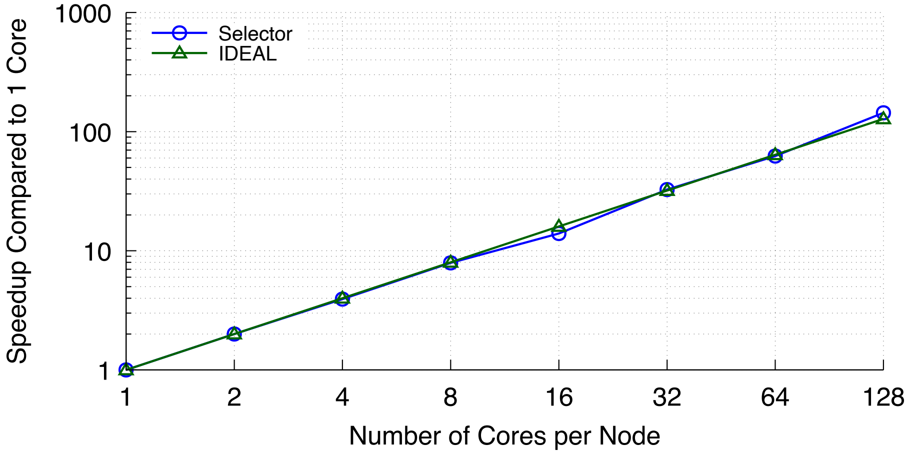
Routers Dataset
Strong Scaling



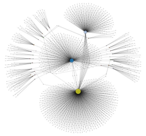


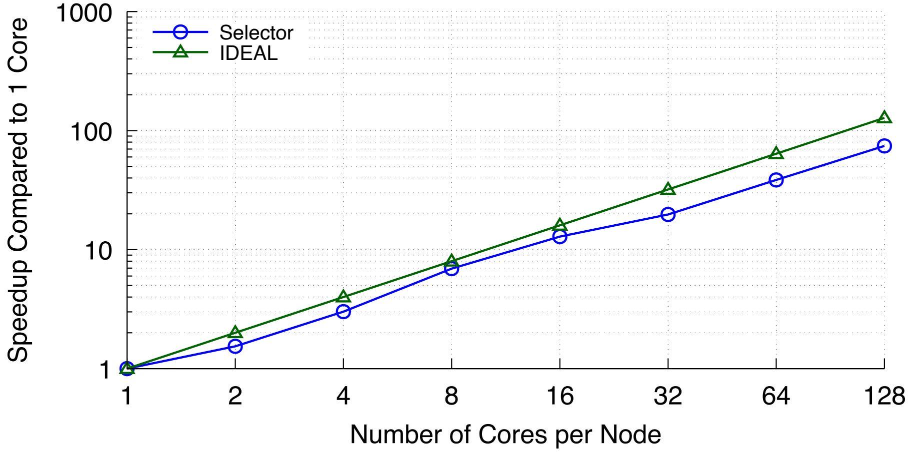
IPs Dataset
Strong Scaling



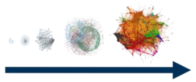


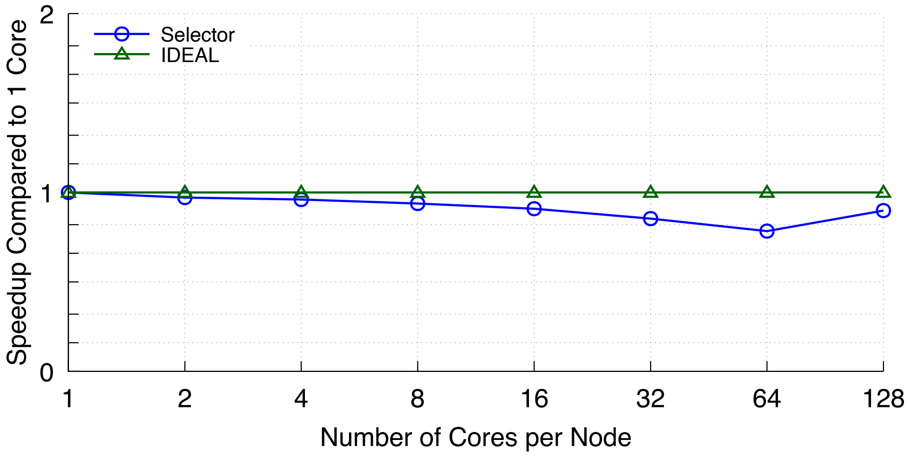
ASes Dataset
Strong Scaling





Synthetic Dataset
Weak Scaling

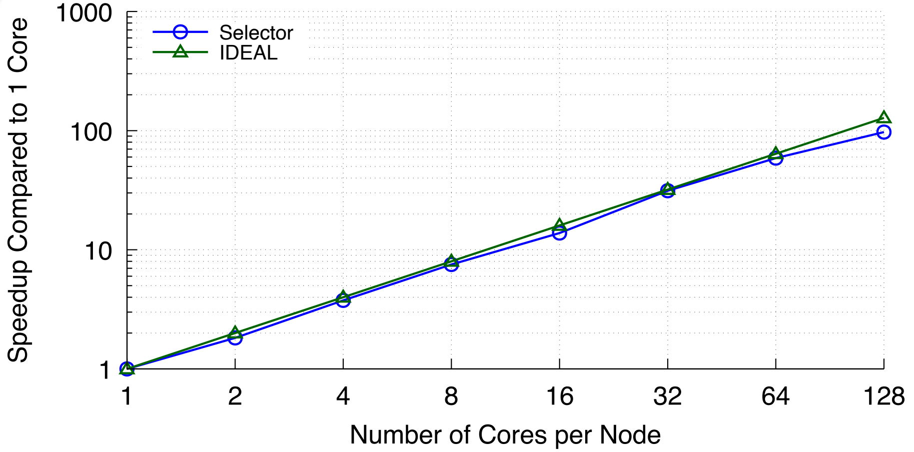




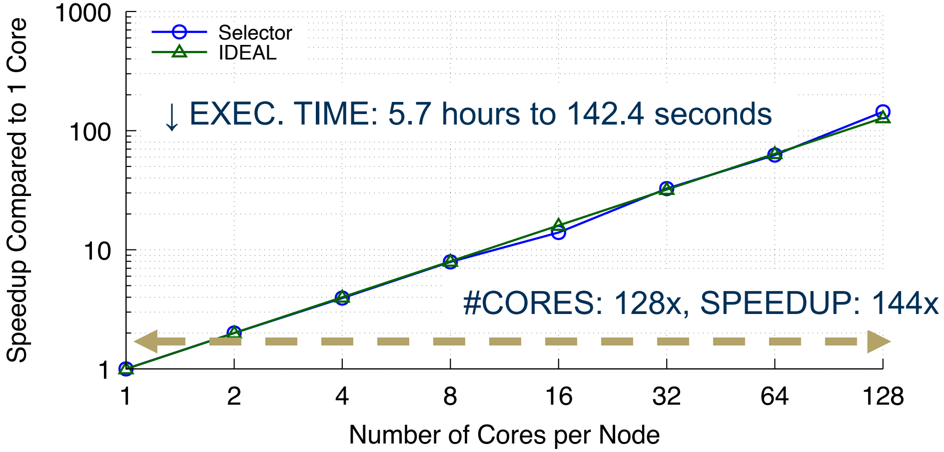
Georgia Tech

Performance Results: Scalability for SCALE-UP

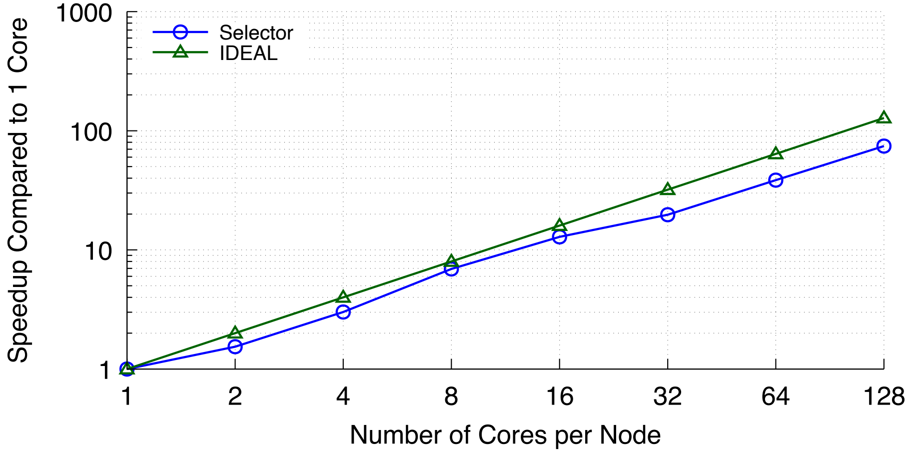
Routers Dataset
Strong Scaling



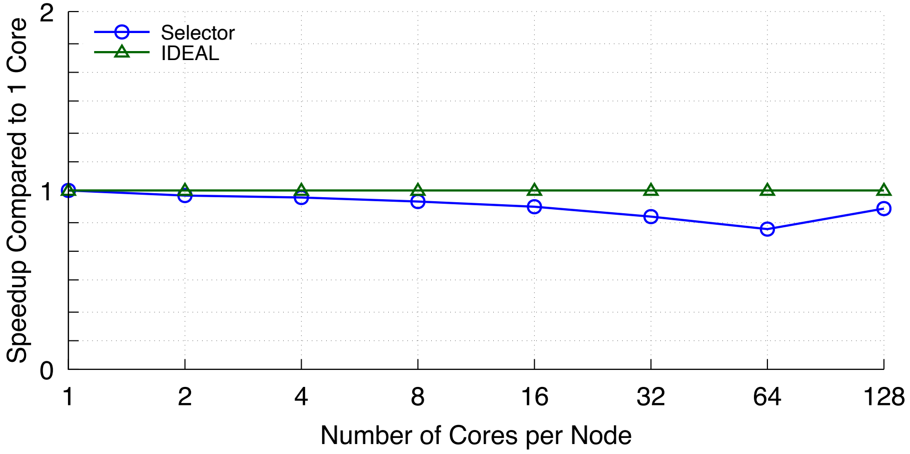
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling

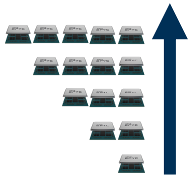


Synthetic Dataset
Weak Scaling

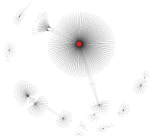


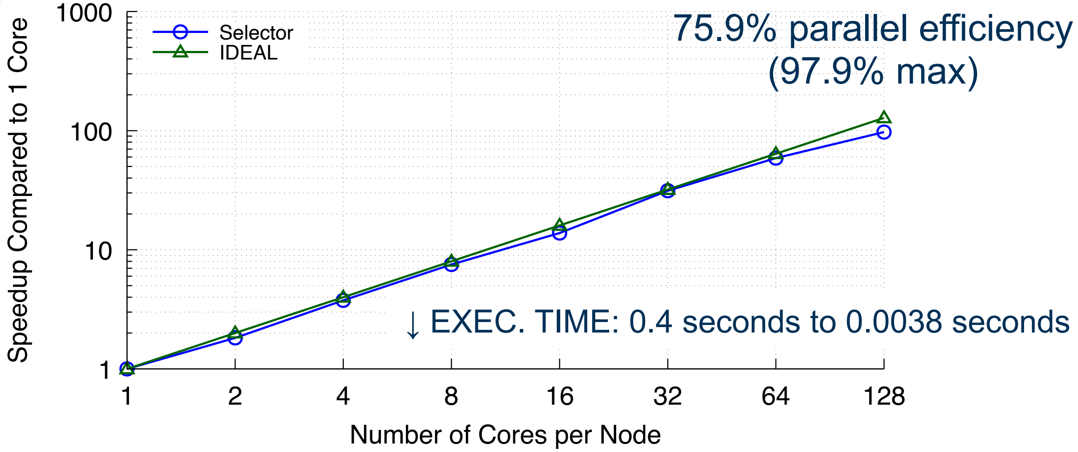
Georgia Tech

Performance Results: Scalability for SCALE-UP

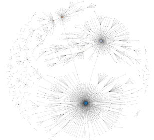


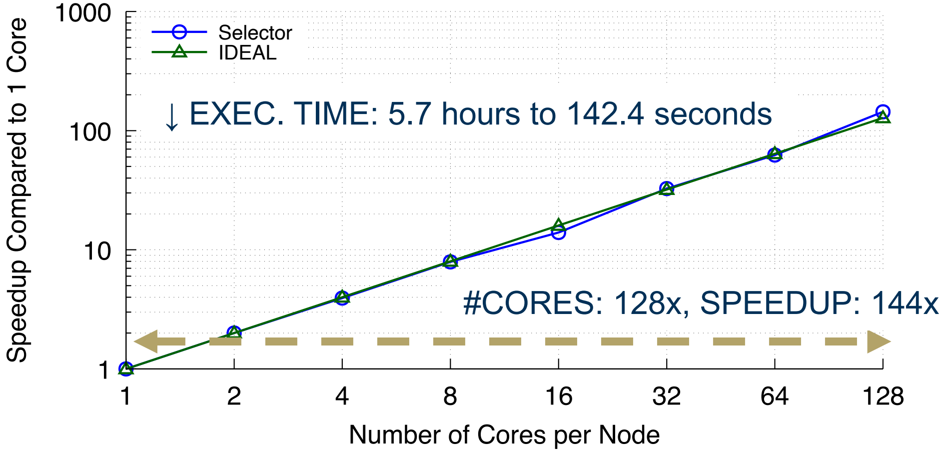
Routers Dataset
Strong Scaling



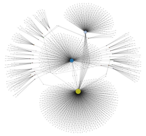


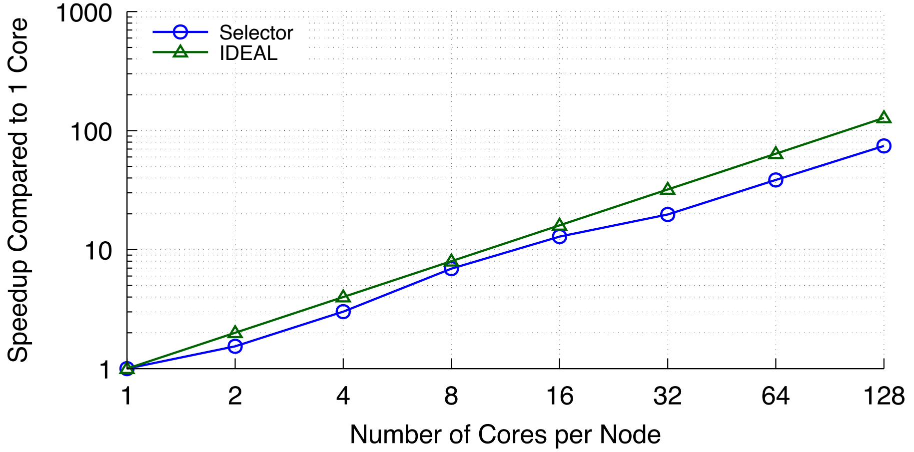
IPs Dataset
Strong Scaling



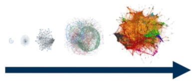


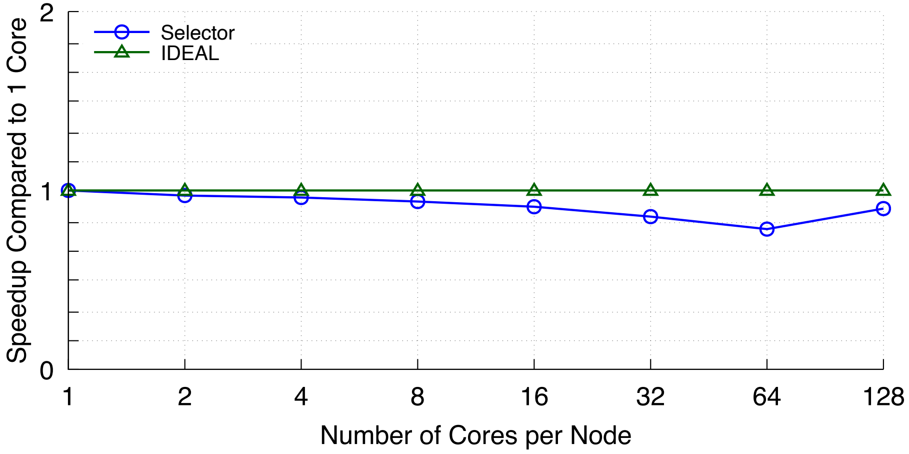
ASes Dataset
Strong Scaling





Synthetic Dataset
Weak Scaling

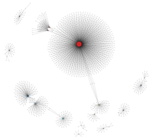


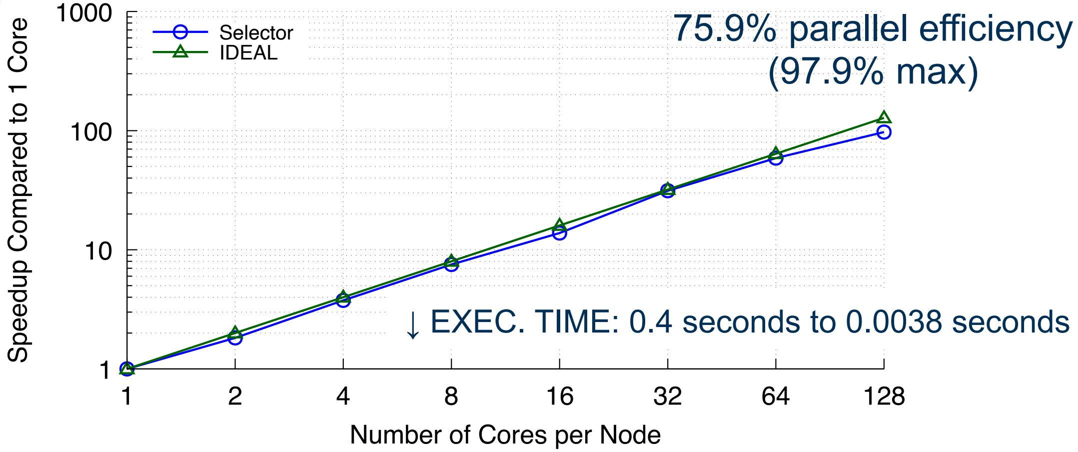


Georgia Tech

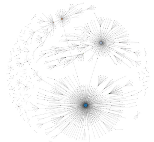
Performance Results: Scalability for SCALE-UP

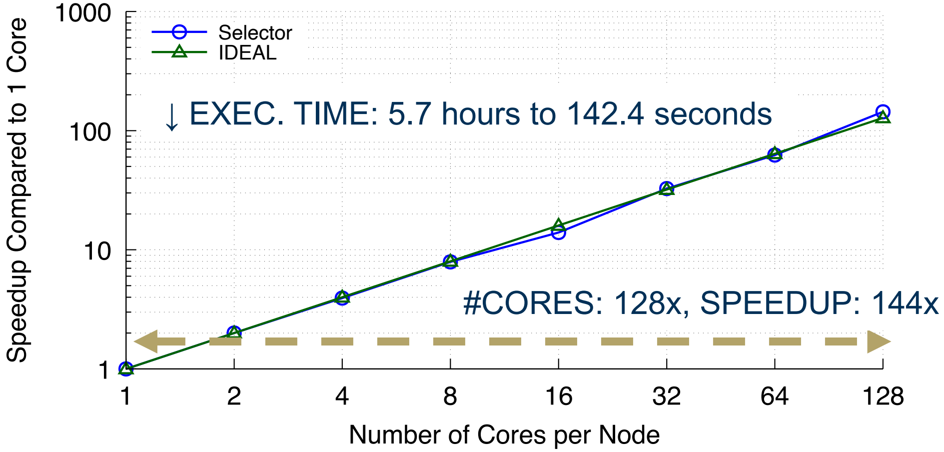
Routers Dataset
Strong Scaling



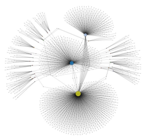


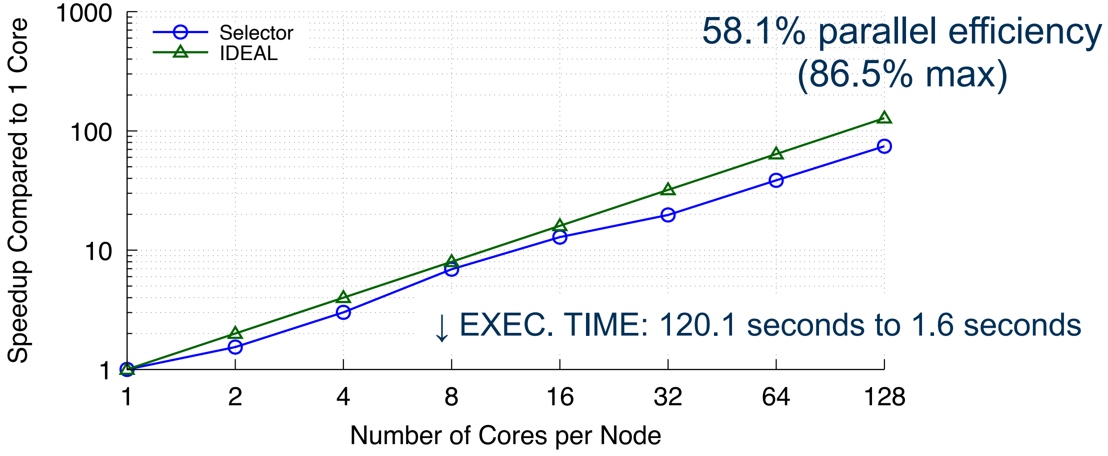
IPs Dataset
Strong Scaling



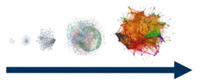


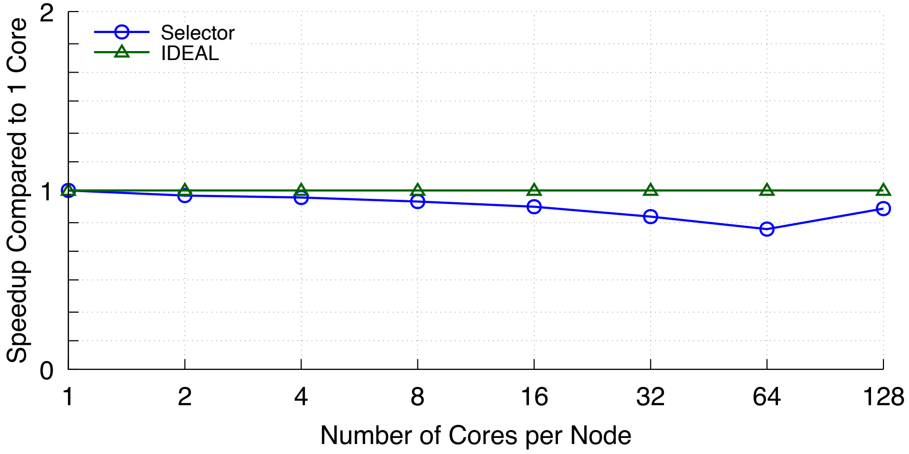
ASes Dataset
Strong Scaling





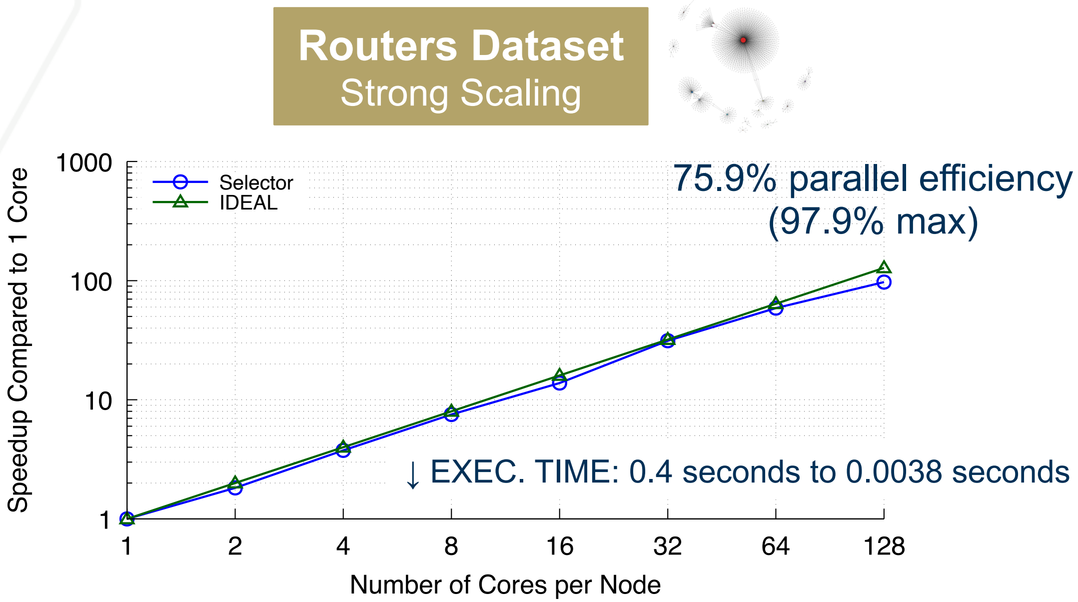
Synthetic Dataset
Weak Scaling

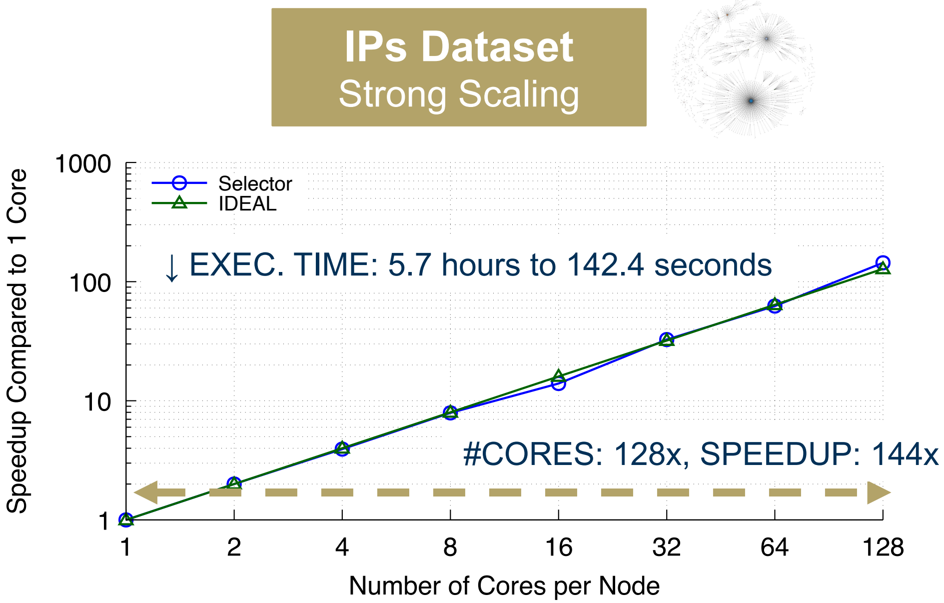


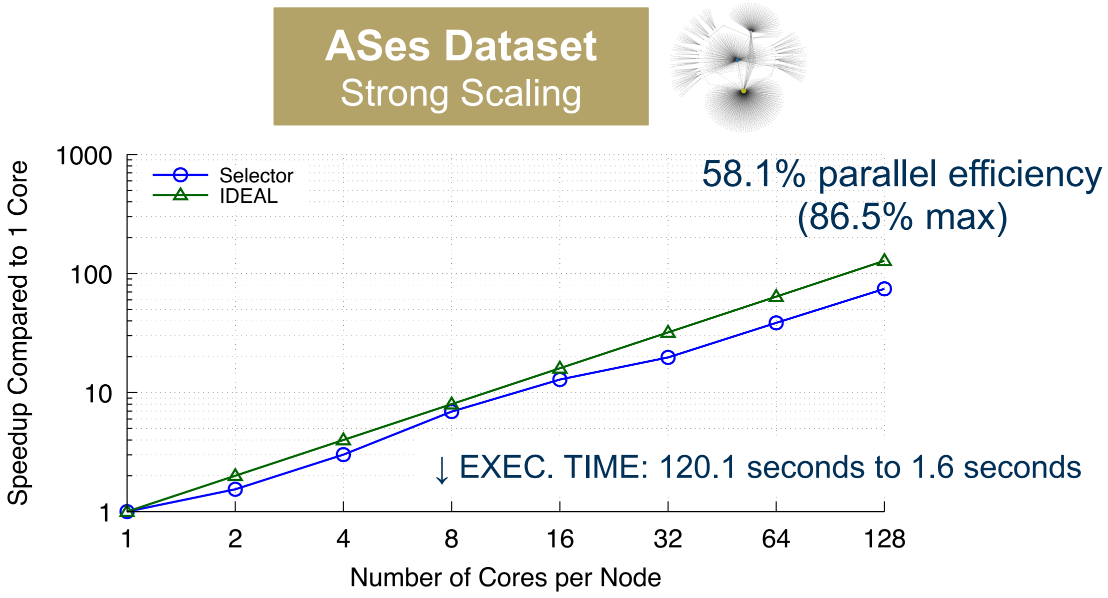


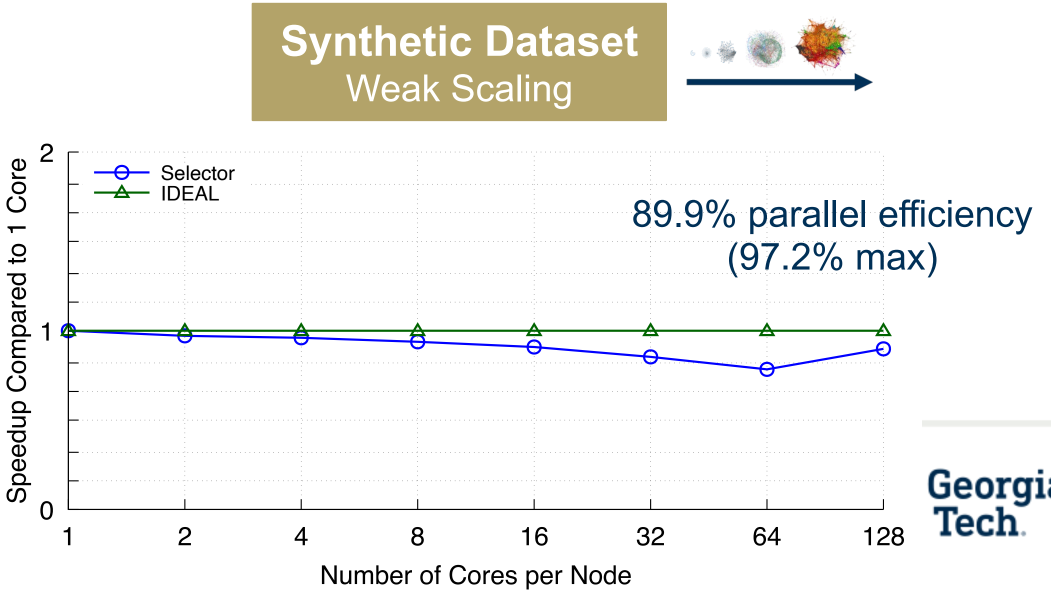
Georgia Tech

Performance Results: Scalability for SCALE-UP









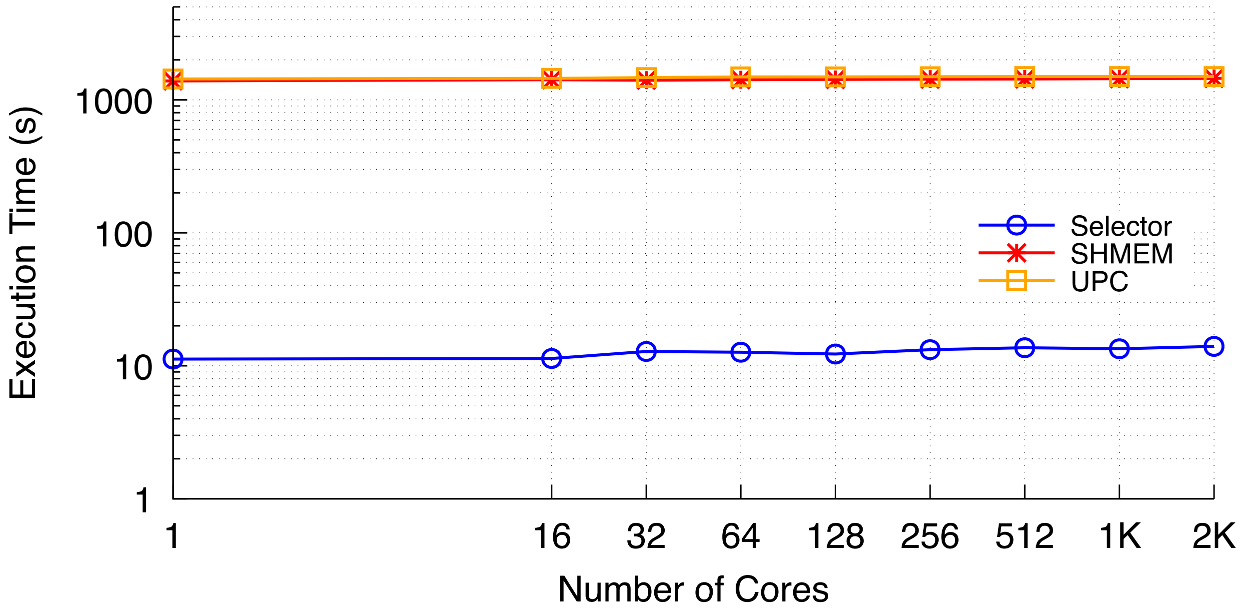
Georgia Tech

Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the `SCALE-OUT` dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC

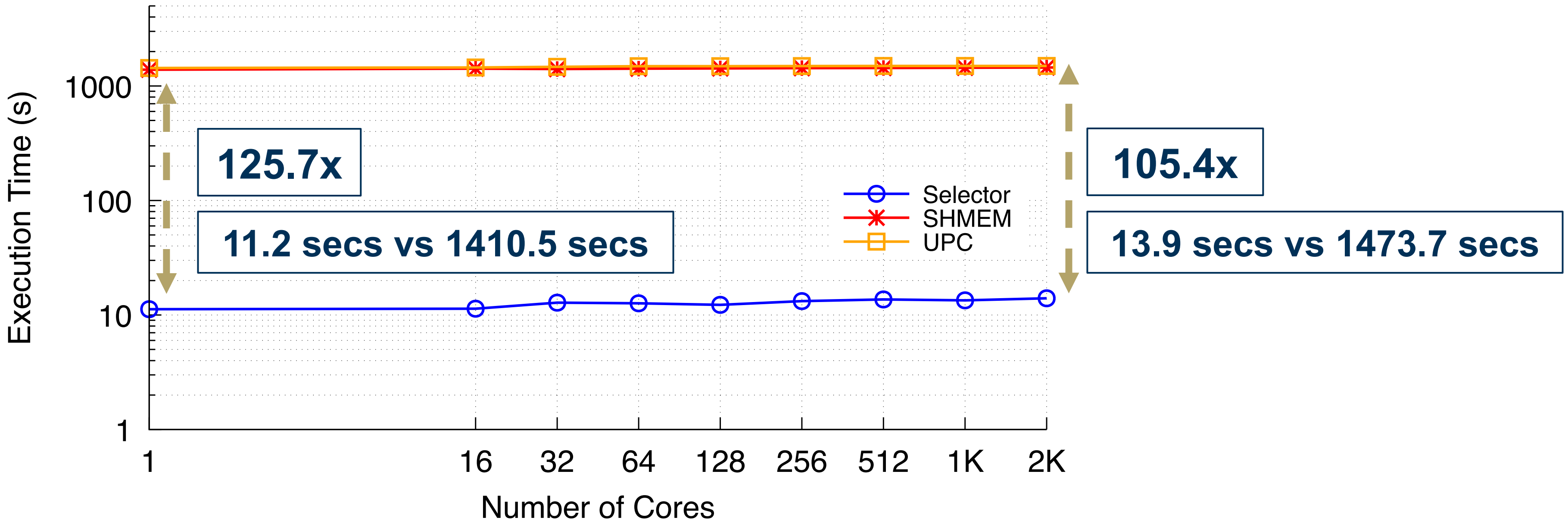
Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the `SCALE-OUT` dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC



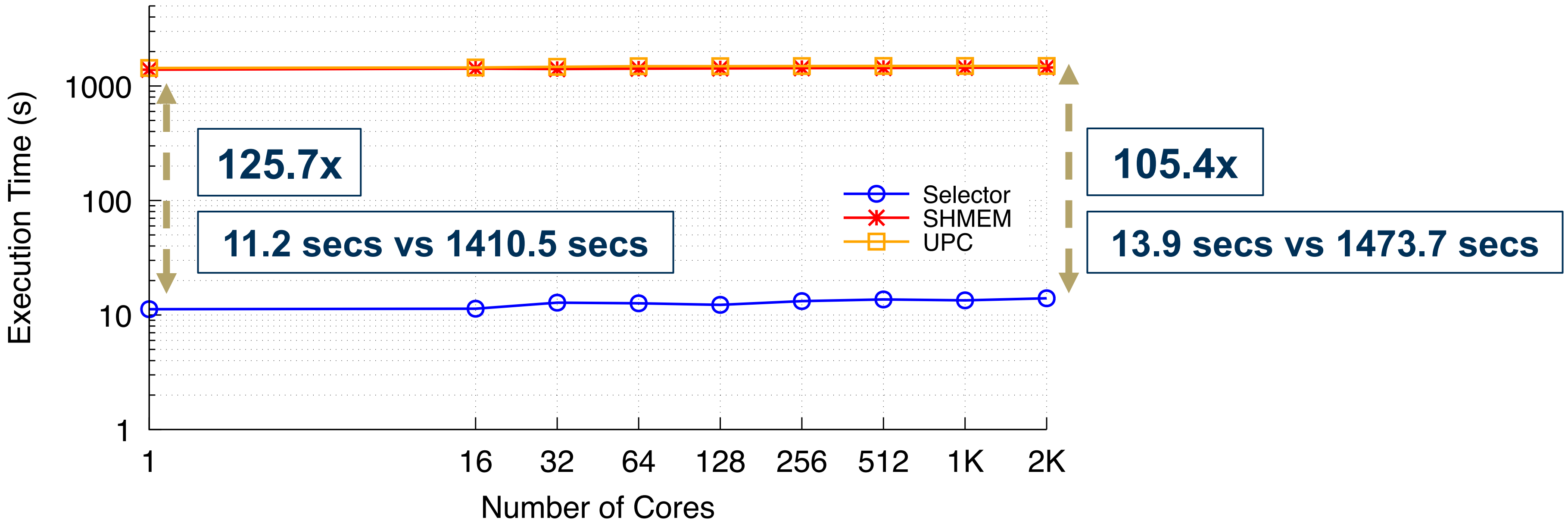
Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the `SCALE-OUT` dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC



Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the `SCALE-OUT` dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC



Attributed to the distributed and async. characteristics of our centrality algorithm as well as the efficient backend runtime

Georgia Tech

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1 **Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources**

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1. Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources
2. Our algorithm allows for effective real-time decision making

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1. Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources
2. Our algorithm allows for effective real-time decision making
3. Our algorithm reduces network energy consumption due to classification of areas of high-traffic, congestion, and underutilized resources

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1. Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources
2. Our algorithm allows for effective real-time decision making
3. Our algorithm reduces network energy consumption due to classification of areas of high-traffic, congestion, and underutilized resources
4. Our algorithm can be applied to internet network topologies within global internet infrastructure, datacenters, cloud, and private networks

Georgia Tech

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1. Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources
2. Our algorithm allows for effective real-time decision making
3. Our algorithm reduces network energy consumption due to classification of areas of high-traffic, congestion, and underutilized resources
4. Our algorithm can be applied to internet network topologies within global internet infrastructure, datacenters, cloud, and private networks
5. Our algorithm can be extended to other domains

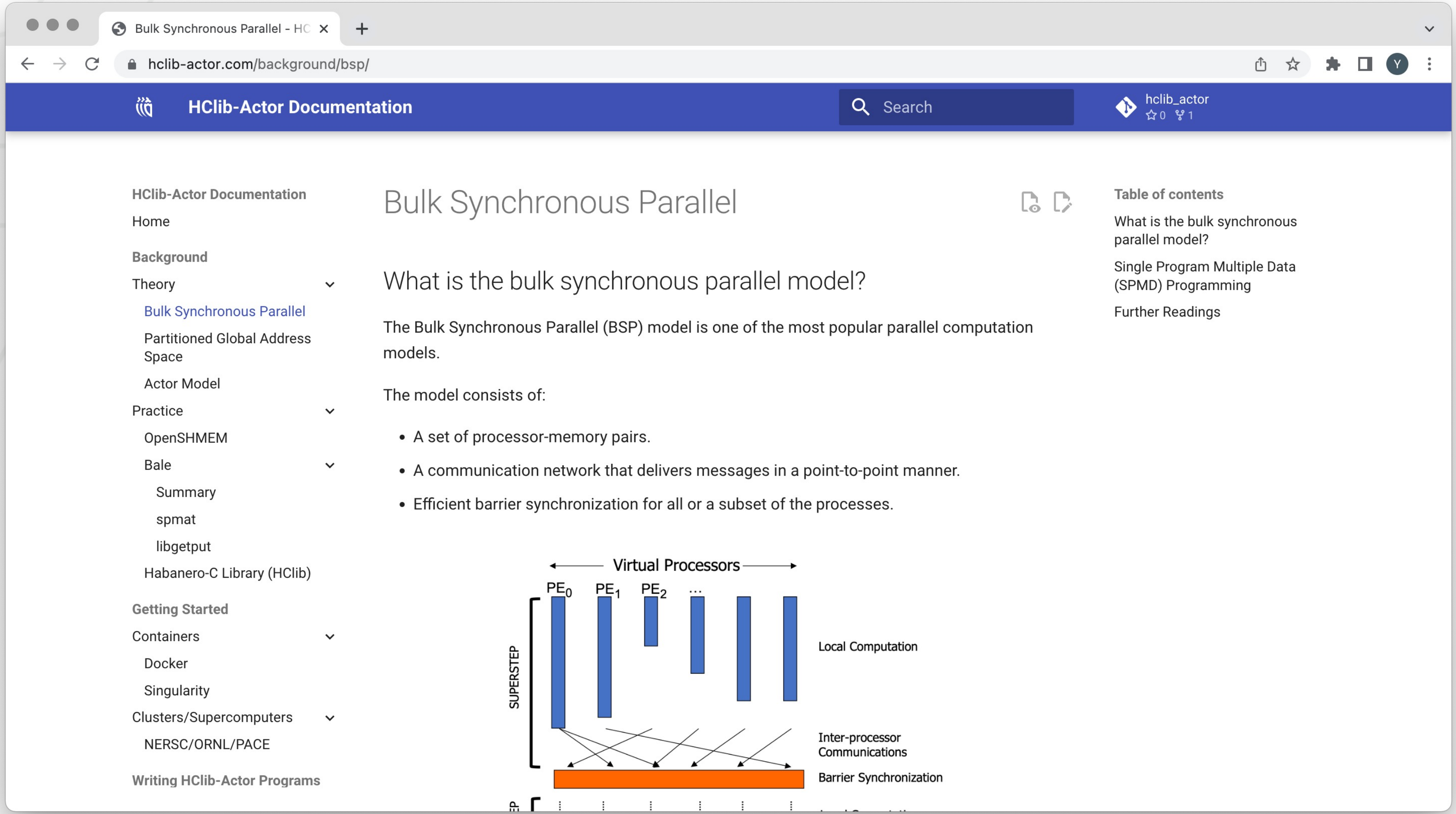
Georgia Tech

DEMO

A Distributed, Asynchronous Algorithm for Large-Scale Internet Network Topology Analysis

Georgia Tech

You can try this at home… Just visit hclib-actor.com !



ACKNOWLEDGEMENT

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through the Advanced Graphical Intelligence Logical Computing Environment (AGILE) research program, under Army Research Office (ARO) contract number W911NF22C0083. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government.

IEEE/ACM CCGRID 2024

The 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

A Distributed, Asynchronous Algorithm for Large-Scale Internet Network Topology Analysis

Youssef Elmougy, Akihiro Hayashi, and Vivek Sarkar

Habanero Extreme Scale Software Research Lab
Georgia Institute of Technology

Thank you for your attention!

Georgia Tech