Fault-Tolerant Distributed Transactions on Blockchain

Beyond the Design of PBFT

Suyash Gupta, Jelle Hellings, Mohammad Sadoghi
Previously: PBFT

Central Question

What is the *expected performance* of PBFT? Motivate!
On the Performance of Consensus

**Consensus throughput**  Decisions per second made by consensus.

**Consensus latency**  Duration of a single round of consensus.

**Resource utilization**  The cost of consensus (e.g., computational, network bandwidth).

*Imbalance* in resource utilized by replicas (e.g., primary).

▶ Low loads: Function of the consensus latency.

▶ High loads: Function of the consensus throughput.
On the Performance of Consensus

**Consensus throughput**  Decisions per second made by consensus.

**Consensus latency**  Duration of a single round of consensus.

**Resource utilization**  The cost of consensus (e.g., computational, network bandwidth). *Imbalance* in resource utilized by replicas (e.g., primary).

**Complexity**  Complexity of normal-case and of recovery (e.g., view-change).

**Failure Model**  The types of failures consensus can deal with.
On the Performance of Consensus

**Consensus throughput**  Decisions per second made by consensus.

**Consensus latency**  Duration of a single round of consensus.

**Resource utilization**  The cost of consensus (e.g., computational, network bandwidth).  
*Imbalance* in resource utilized by replicas (e.g., primary).

**Complexity**  Complexity of normal-case and of recovery (e.g., view-change).

**Failure Model**  The types of failures consensus can deal with.

**Client latency**  Duration between a client request and the outcome.
On the Performance of Consensus

Consensus throughput: Decisions per second made by consensus.
Consensus latency: Duration of a single round of consensus.
Resource utilization: The cost of consensus (e.g., computational, network bandwidth).
   *Imbalance* in resource utilized by replicas (e.g., primary).

Complexity: Complexity of normal-case and of recovery (e.g., view-change).
Failure Model: The types of failures consensus can deal with.

Client latency: Duration between a client request and the outcome.
   - *Low loads*: Function of the consensus latency.
   - *High loads*: Function of the consensus throughput.
Determining the Performance Variables

Number of replicas determines the amount of messages exchanged.
Network bandwidth determines how long it takes to exchange these messages.
Message delay determines how long it takes for sent messages to arrive.
Computational speed determines the speed by which messages are processed.
Determining the Performance Variables

- **Number of replicas**: determines the amount of messages exchanged.
- **Network bandwidth**: determines how long it takes to exchange these messages.
- **Message delay**: determines how long it takes for sent messages to arrive.
- **Computational speed**: determines the speed by which messages are processed.

System processing client transactions
Determining the Performance Variables

- **Number of replicas** determines the amount of messages exchanged.
- **Network bandwidth** determines how long it takes to exchange these messages.
- **Message delay** determines how long it takes for sent messages to arrive.
- **Computational speed** determines the speed by which messages are processed.

System processing client transactions

- Bottlenecks *outside consensus*: speed by which replicas *execute transactions*. 
Determining the Performance Variables

- **Number of replicas** determines the amount of messages exchanged.
- **Network bandwidth** determines how long it takes to exchange these messages.
- **Message delay** determines how long it takes for sent messages to arrive.
- **Computational speed** determines the speed by which messages are processed.

System processing client transactions

- Bottlenecks *outside consensus*: speed by which replicas *execute transactions*.
- Computational speed typically sufficient when *parallelization* is used.
Determining the Performance Variables

Number of replicas determines the amount of messages exchanged. Network bandwidth determines how long it takes to exchange these messages. Message delay determines how long it takes for sent messages to arrive. Computational speed determines the speed by which messages are processed.

System processing client transactions

▪ Bottlenecks outside consensus: speed by which replicas execute transactions.
▪ Computational speed typically sufficient when parallelization is used.

Bottleneck in practice: consensus performance in terms of throughput and latency (as a function of network bandwidth and message delay).
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each.
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each.

$\triangleright$ $n - 1$ messages

$\triangleright$ $s_t$ B each
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each.

\[
\begin{align*}
\text{Propose: } & \quad (n-1) s_t = 3 \cdot 4048 = 12144 \text{ B} \\
\text{Prepare: } & \quad 2(n-1) s_m = 4 \cdot 256 = 1024 \text{ B} \\
\text{Commit: } & \quad \delta = 0.015 \text{ s}.
\end{align*}
\]
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each}$.

$$
\begin{align*}
\text{Propose:} & \quad s_t = 4048 \text{ B each.} \\
\text{Last byte arrives after} & \quad \delta = 15 \text{ ms.}
\end{align*}
$$

$$
\begin{align*}
\text{Propose} & \quad \text{n - 1 messages} \\
\text{Prepare} & \quad s_t \text{ B each} \\
\text{Commit} & \quad B \text{ MiB/s} \\
\text{Last byte} & \quad \text{arrives after} \quad \delta
\end{align*}
$$

$$
\begin{align*}
(n - 1)s_t & = 3 \cdot 4048 = 12144 \text{ B}
\end{align*}
$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth \( B = 100 \text{ MiB/s} \) and delay \( \delta = 15 \text{ ms} \)

Propose: \( s_t = 4048 \text{ B each} \).

\[
\begin{align*}
\text{Propose} & : n-1 \text{ messages} \\
\text{Prepare} & : s_t \text{ B each} \\
\text{Commit} & : B \text{ MiB/s} \\
\text{Last byte arrives after } \delta & : \frac{(n-1)s_t}{B} + \delta = \frac{12144}{100 \cdot 2^{20}} + 0.015 \approx 0.0151 \text{ s.}
\end{align*}
\]
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each}$. Prepare and Commit: $s_m = 256 \text{ B each}$.

- $n - 1$ messages
- $s_m$ B each
- $B$ MiB/s
- Last byte arrives after $\delta$

Diagram showing the process with $n$ nodes and delays and bandwidths.
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each}$
Prepare and Commit: $s_m = 256 \text{ B each}$

$$\frac{(n-1)s_m}{B} + \delta = \frac{768}{100 \cdot 2^{20}} + 0.015 \approx 0.0150 \text{ s.}$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each. Prepare and Commit: $s_m = 256$ B each.

- $n - 1$ messages
- $s_m$ B each
- $B$ MiB/s
- Last byte arrives after $\delta$

\[ \Delta_{PBFT} = (n - 1) s_t B + \delta + (n - 1) s_m B + 3 \delta \approx 3 \delta \text{ (assuming high delay relative to bandwidth)} \]
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each. Prepare and Commit: $s_m = 256$ B each.
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each}$. Prepare and Commit: $s_m = 256 \text{ B each}$.

$$\Delta_{\text{PBFT}} = \frac{(n - 1)s_t}{B}$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each.}$ Prepare and Commit: $s_m = 256 \text{ B each.}$

$$\Delta_{PBFT} = \frac{(n-1)s_t}{B} + \delta$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each. Prepare and Commit: $s_m = 256$ B each.

$$\Delta_{PBFT} = \frac{(n - 1)s_t}{B} + \delta + \frac{(n - 1)s_m}{B}$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each}$. Prepare and Commit: $s_m = 256 \text{ B each}$.

\[
\Delta_{\text{PBFT}} = \frac{\left(n - 1\right)s_t}{B} + \delta + \frac{\left(n - 1\right)s_m}{B} + \delta
\]
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100 \text{ MiB/s}$ and delay $\delta = 15 \text{ ms}$

Propose: $s_t = 4048 \text{ B each}$. Prepare and Commit: $s_m = 256 \text{ B each}$.

$$\Delta_{\text{PBFT}} = \left( \frac{n - 1}{B} \right) s_t + \delta + \left( \frac{n - 1}{B} \right) s_m + \delta + \left( \frac{n - 1}{B} \right) s_m$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each. Prepare and Commit: $s_m = 256$ B each.

\[
\Delta_{\text{PBFT}} = \frac{(n-1)s_t}{B} + \delta + \frac{(n-1)s_m}{B} + \delta + \frac{(n-1)s_m}{B} + \delta
\]
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each. Prepare and Commit: $s_m = 256$ B each.

$$\Delta_{PBFT} = \frac{(n - 1)s_t}{B} + \delta + \frac{(n - 1)s_m}{B} + \delta + \frac{(n - 1)s_m}{B} + \delta$$

$$= \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta$$
The Single-Round Cost of PBFT (Sketch)

Assumption: Network bandwidth $B = 100$ MiB/s and delay $\delta = 15$ ms

Propose: $s_t = 4048$ B each. Prepare and Commit: $s_m = 256$ B each.

$$\Delta_{PBFT} = \frac{(n-1)s_t}{B} + \delta + \frac{(n-1)s_m}{B} + \delta + \frac{(n-1)s_m}{B} + \delta$$

$$= \frac{(n-1)s_t + 2(n-1)s_m}{B} + 3\delta$$

$$\approx 3\delta$$ (assuming high delay relative to bandwidth).
The Throughput of PBFT

Sequential: Next consensus round starts after finishing the current round

\[ T_{\text{PBFT}} = \frac{1}{\Delta_{\text{PBFT}}} = \frac{B}{(n - 1)s_t + 2(n - 1)s_m + 3B\delta}. \]
The Throughput of PBFT

Sequential: Next consensus round starts after finishing the current round

\[ T_{PBFT} = \frac{1}{\Delta_{PBFT}} = \frac{B}{(n - 1)s_t + 2(n - 1)s_m + 3B\delta}. \]

Assumption: \( B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)

\( (\delta = 15 \text{ ms}) \)
The Throughput of PBFT

Sequential: Next consensus round starts after finishing the current round

\[
T_{\text{PBFT}} = \frac{1}{\Delta_{\text{PBFT}}} = \frac{B}{(n - 1)s_t + 2(n - 1)s_m + 3B\delta}.
\]

Assumption: \( B = 100 \text{ MiB/s} \), \( s_t = 4048 \text{ B} \), \( s_m = 256 \text{ B} \)

\((\delta = 15 \text{ ms})\)

![Graph of Throughput vs. Number of Replicas](image1)

![Graph of Throughput vs. Message Delay](image2)
Implementation techniques for PBFT

Realistic wide-area message delays: 10 ms–300 ms

The throughput $T_{PBFT}$ of sequential PBFT is *impractically low*. 
Implementation techniques for PBFT

Realistic wide-area message delays: 10 ms–300 ms

The throughput $T_{PBFT}$ of sequential PBFT is *impractically low*.

Fine-tuning PBFT implementations

- **Batching** many transactions per consensus decision.
- **Out-of-order processing** many consensus decisions at the same time.
- **Overlapping** phases of consecutive rounds.
## Batching Client Requests

The cost of a single round of PBFT

<table>
<thead>
<tr>
<th>Message</th>
<th>Sent by</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propose</td>
<td>Primary</td>
<td>$s_t$</td>
</tr>
<tr>
<td>Prepare</td>
<td>Backups</td>
<td>$s_m$</td>
</tr>
<tr>
<td>Commit</td>
<td>All</td>
<td>$s_m$</td>
</tr>
</tbody>
</table>
## Batching Client Requests

### The cost of a single round of PBFT

Batching: each decision is on $m$ transactions.

<table>
<thead>
<tr>
<th>Message</th>
<th>Sent by</th>
<th>Size</th>
<th>$(\text{batch})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propose</td>
<td>Primary</td>
<td>$s_t$</td>
<td>$ms_t$</td>
</tr>
<tr>
<td>Prepare</td>
<td>Backups</td>
<td>$s_m$</td>
<td>$s_m$</td>
</tr>
<tr>
<td>Commit</td>
<td>All</td>
<td>$s_m$</td>
<td>$s_m$</td>
</tr>
</tbody>
</table>
Batching Client Requests

The cost of a single round of PBFT

Batching: each decision is on \( m \) transactions.

<table>
<thead>
<tr>
<th>Message</th>
<th>Sent by</th>
<th>Size</th>
<th>((\text{batch}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propose</td>
<td>Primary</td>
<td>( s_t )</td>
<td>( m s_t )</td>
</tr>
<tr>
<td>Prepare</td>
<td>Backups</td>
<td>( s_m )</td>
<td>( s_m )</td>
</tr>
<tr>
<td>Commit</td>
<td>All</td>
<td>( s_m )</td>
<td>( s_m )</td>
</tr>
</tbody>
</table>

Total: \( 2n(n - 1) \quad O(s_t n + s_m n^2) \quad O(m s_t n + s_m n^2) \)
The Throughput of PBFT with Batching

Sequential, batching $m$ transactions per consensus round

$$\Delta_{\text{PBFT}-m} = \frac{m(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta;$$
The Throughput of PBFT with Batching

Sequential, batching $m$ transactions per consensus round

$$\Delta_{PBFT-m} = \frac{m(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta; \quad T_{PBFT-m} = m \frac{1}{\Delta_{PBFT-m}}.$$
The Throughput of PBFT with Batching

Sequential, batching $m$ transactions per consensus round

$$\Delta_{PBFT-m} = \frac{m(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta; \quad T_{PBFT-m} = m\frac{1}{\Delta_{PBFT-m}}.$$ 

Assumption: $B = 100$ MiB/s, $\delta = 15$ ms, $s_t = 4048$ B, $s_m = 256$ B
The Throughput of PBFT with Batching

Sequential, batching \( m \) transactions per consensus round

\[
\Delta_{\text{PBFT}-m} = \frac{m(n-1)s_t + 2(n-1)s_m}{B} + 3\delta; \quad T_{\text{PBFT}-m} = m\frac{1}{\Delta_{\text{PBFT}-m}}.
\]

Assumption: \( B = 100 \text{ MiB/s}, \delta = 15 \text{ ms}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)
The Throughput of PBFT with Batching

Sequential, batching \( m \) transactions per consensus round

\[
\Delta_{\text{PBFT-}m} = \frac{m(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta;
\]

\[
T_{\text{PBFT-}m} = m\frac{1}{\Delta_{\text{PBFT-}m}}.
\]

Assumption: \( B = 100 \text{ MiB/s}, \delta = 15 \text{ ms}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)
The Throughput of PBFT with Batching

Sequential, batching $m$ transactions per consensus round

$$\Delta_{\text{PBFT}-m} = \frac{m(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta; \quad T_{\text{PBFT}-m} = m\frac{1}{\Delta_{\text{PBFT}-m}}.$$ 

Assumption: $B = 100 \text{ MiB/s}$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
The Throughput of PBFT with Batching

Sequential, batching $m$ transactions per consensus round

$$\Delta_{\text{PBFT}-m} = m\left(\frac{n - 1}{2(n - 1)}s_t + 2\frac{(n - 1)}{2(n - 1)}s_m\right) + 3\delta;$$

$$T_{\text{PBFT}-m} = m\frac{1}{\Delta_{\text{PBFT}-m}}.$$

Assumption: $B = 100 \text{ MiB/s}$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$

![Consensus Latency (ms) vs Number of Replicas](chart1)

![Throughput (round/s) vs Number of Replicas](chart2)
Using Batching to Improve Throughput Scalability

<table>
<thead>
<tr>
<th></th>
<th>Messages (per trans.)</th>
<th>Size (per trans.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$2n(n - 1)$</td>
<td>$O(s_t n + s_m n^2)$</td>
</tr>
</tbody>
</table>
Using Batching to Improve Throughput Scalability

<table>
<thead>
<tr>
<th></th>
<th>Messages (per trans.)</th>
<th>Size (per trans.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$2n(n - 1)$</td>
<td>$O(s_t n + s_m n^2)$</td>
</tr>
</tbody>
</table>

Assumption: $B = 100 \text{ MiB}/s$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
Using Batching to Improve Throughput Scalability

<table>
<thead>
<tr>
<th></th>
<th>Messages (per trans.)</th>
<th>Size (per trans.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$2n(n - 1)$</td>
<td>$O(s_t n + s_m n^2)$</td>
<td>$O(s_t n + s_m n^2)$</td>
</tr>
<tr>
<td>PBFT-n</td>
<td>$2n(n - 1)$</td>
<td>$2(n - 1)$</td>
<td>$O(n s_t n + s_m n^2)$</td>
</tr>
</tbody>
</table>

Assumption:

$B = 100 \text{ MiB/s}$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
### Using Batching to Improve Throughput Scalability

<table>
<thead>
<tr>
<th></th>
<th>Messages (per trans.)</th>
<th>Size (per trans.)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$2n(n - 1)$</td>
<td>$O(s_t n + s_m n^2)$</td>
<td>$O(s_t n + s_m n^2)$</td>
</tr>
<tr>
<td>PBFT-n</td>
<td>$2n(n - 1)$</td>
<td>$2(n - 1)$</td>
<td>$O(ns_t n + s_m n^2)$</td>
</tr>
</tbody>
</table>

Assumption: $B = 100$ MiB/s, $\delta = 15$ ms, $s_t = 4048$ B, $s_m = 256$ B
Using Batching to Improve Throughput Scalability

<table>
<thead>
<tr>
<th></th>
<th>Messages (per trans.)</th>
<th>Size (per trans.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBFT</td>
<td>$2n(n-1)$</td>
<td>$O(s_t n + s_m n^2)$</td>
</tr>
<tr>
<td>PBFT-n</td>
<td>$2n(n-1)$</td>
<td>$O(n s_t n + s_m n^2)$</td>
</tr>
</tbody>
</table>

Assumption: $B = 100 \text{ MiB/s}$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
Resource Utilization of Sequential PBFT

Assumption: $n = 4$, $B = 100 \text{ MiB/s}$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$

$$\Delta_{\text{PBFT}} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta \approx 45.1 \text{ ms}.$$
Resource Utilization of Sequential PBFT

Assumption: \( n = 4, B = 100 \text{ MiB/s}, \delta = 15 \text{ ms}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)

\[
\Delta_{\text{PBFT}} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta \approx 45.1 \text{ ms}.
\]

\( \Delta_{\text{PBFT}} \) consists of two components:

- **Message transfer**
  \[
  \approx 0.1 \text{ ms}
  \]

- **Waiting**
  \[
  \approx 45.1 \text{ ms}
  \]
Resource Utilization of Sequential PBFT

Assumption: $n = 4$, $B = 100 \text{MiB/s}$, $\delta = 15 \text{ms}$, $s_t = 4048 \text{B}$, $s_m = 256 \text{B}$

$$\Delta_{PBFT} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta \approx 45.1 \text{ms}.$$
Resource Utilization of Sequential PBFT

Assumption: $n = 4, B = 100 \text{ MiB/s}, \delta = 15 \text{ ms}, s_t = 4048 \text{ B}, s_m = 256 \text{ B}$

$$\Delta_{PBFT} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta \approx 45.1 \text{ ms}.$$
Resource Utilization of Sequential PBFT

Assumption: $n = 4$, $B = 100$ MiB/s, $\delta = 15$ ms, $s_t = 4048$ B, $s_m = 256$ B

$$\Delta_{PBFT} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta \approx 45.1 \text{ ms.}$$

- message transfer $\approx 0.1$ ms
- waiting $45.0$ ms

![Graph showing bandwidth utilization at the primary over message delay](image-url)
Resource Utilization of Sequential PBFT

Assumption: \( n = 4, B = 100 \text{ MiB/s}, \delta = 15 \text{ ms}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)

\[
\Delta_{\text{PBFT}} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta \approx 45.1 \text{ ms}.
\]

- **message transfer**: \( \approx 0.1 \text{ ms} \)
- **waiting**: \( 45.0 \text{ ms} \)

![Bandwidth Utilization at the Primary](image)

\( \delta = 15 \text{ ms} \)
Problem: Resource Utilization of PBFT

- Typically less than 5% bandwidth utilization at primary.
- With huge batches still less than 40% bandwidth utilization at primary.
Problem: Resource Utilization of PBFT

- Typically less than 5% bandwidth utilization at primary.
- With huge batches still less than 40% bandwidth utilization at primary.

To maximize throughput: use all bandwidth at the primary.
Problem: Resource Utilization of PBFT

- Typically *less than 5%* bandwidth utilization at primary.
- With huge batches still *less than 40%* bandwidth utilization at primary.

To maximize throughput: use *all* bandwidth at the primary.

Out-of-order processing

Primary can proposes *future rounds* before current rounds are finished.
Problem: Resource Utilization of PBFT

- Typically *less than 5%* bandwidth utilization at primary.
- With huge batches still *less than 40%* bandwidth utilization at primary.

To maximize throughput: use *all* bandwidth at the primary.

Out-of-order processing
Primary can proposes *future rounds* before current rounds are finished.

Practical challenges
- Memory usage: replicas maintain meta-data for each *active* round.
- Byzantine behavior: exhaust the set of round numbers.
Problem: Resource Utilization of PBFT

- Typically less than 5% bandwidth utilization at primary.
- With huge batches still less than 40% bandwidth utilization at primary.

To maximize throughput: use all bandwidth at the primary.

Out-of-order processing
Primary can proposes future rounds before current rounds are finished.

Practical challenges
- Memory usage: replicas maintain meta-data for each active round.
- Byzantine behavior: exhaust the set of round numbers.

Limit proposals to an active window of valid rounds.
E.g., only proposals in 1000 rounds after the last finished round.
The Single-Round Cost of PBFT (revised)

Assumption: Primary does most work ($s_t > s_m$)
The Single-Round Cost of PBFT (revised)

Assumption: Primary does most work \((s_t > s_m)\)

\[
\begin{align*}
\text{Propose} & : \quad \text{Send } n - 1 \text{ messages} \\
\text{Prepare} & : \quad s_t \text{ B each} \\
\text{Commit} & : \quad s_t B \text{ each}
\end{align*}
\]
The Single-Round Cost of PBFT (revised)

Assumption: Primary does most work ($s_t > s_m$)

- Receive $n - 1$ messages
- $s_m$ B each
The Single-Round Cost of PBFT (revised)

Assumption: Primary does most work \((s_t > s_m)\)

- Propose
- Prepare
- Commit

- Send and receive \(n - 1\) messages
- \(s_m\) B each
The Single-Round Cost of PBFT (revised)

Assumption: Primary does most work \((s_t > s_m)\)

\[(n - 1)s_t + (n - 1)s_m + 2(n - 1)s_m = (n - 1)(s_t + 3s_m) \text{ B/round.}\]
The Out-of-Order Throughput of PBFT

Assumption: Primary does most work \((s_t > s_m)\)

\[
T_{\text{ooo-PBFT}} = \frac{B}{(n - 1)(s_t + 3s_m)}.
\]

Assumption: \(B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B}\)

![Graph showing throughput vs. number of replicas and message delay](image-url)
The Out-of-Order Throughput of PBFT

Assumption: Primary does most work \((s_t > s_m)\)

\[
T_{\text{ooo-PBFT}} = \frac{B}{(n - 1)(s_t + 3s_m)}.
\]

Assumption: \(B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B}\)
The Out-of-Order Throughput of PBFT

Assumption: Primary does most work ($s_t > s_m$)

$T_{\text{o0o-PBFT}} = \frac{B}{(n - 1)(s_t + 3s_m)}; \quad T_{\text{o0o-PBFT-m}} = \frac{mB}{(n - 1)(ms_t + 3s_m)}.$

Assumption: $B = 100 \text{ MiB/s}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
Overlapping Communication Phases

Out-of-order processing is *complex* to implement.

- Consider a backup replica $r$.
  - Last step of round $\rho$: Commit messages.
  - First step of round $\rho + 1$: Prepare messages.
  
  Idea: Overlapping communication phases
  Merge Commit message of $\rho$ with the Prepare message of $\rho + 1$.
  
  - Make proposal of round $\rho + 1$ refer to round $\rho$.
  - Prepare for round $\rho + 1$ implies Commit for round $\rho$.
  - Primary proposes round $\rho + 1$ after it finished the prepare-phase for round $\rho$.

Implies strict consecutive processing of rounds
Overlapping cannot be combined with out-of-order processing!
Overlapping Communication Phases

Out-of-order processing is *complex* to implement.

Consider a backup replica $R$.

- Last step of round $\rho$: Commit messages.
- First step of round $\rho + 1$: Prepare messages.
Overlapping Communication Phases

Out-of-order processing is *complex* to implement.

Consider a backup replica $R$.

- Last step of round $\rho$: Commit messages.
- First step of round $\rho + 1$: Prepare messages.

**Idea: Overlapping communication phases**

Merge Commit message of $\rho$ with the Prepare message of $\rho + 1$. 
Overlapping Communication Phases

Out-of-order processing is *complex* to implement.

Consider a backup replica $r$.

- Last step of round $\rho$: Commit messages.
- First step of round $\rho + 1$: Prepare messages.

**Idea: Overlapping communication phases**

Merge Commit message of $\rho$ with the Prepare message of $\rho + 1$.

- Make proposal of round $\rho + 1$ *refer* to round $\rho$.
- Prepare for round $\rho + 1$ *implies* Commit for round $\rho$.
- Primary proposes round $\rho + 1$ *after* it finished the prepare-phase for round $\rho$. 

**Implies strict consecutive processing of rounds**

Overlapping cannot be combined with out-of-order processing!
Overlapping Communication Phases

Out-of-order processing is *complex* to implement.

Consider a backup replica $r$.

- Last step of round $\rho$: Commit messages.
- First step of round $\rho + 1$: Prepare messages.

**Idea: Overlapping communication phases**

Merge Commit message of $\rho$ with the Prepare message of $\rho + 1$.

- Make proposal of round $\rho + 1$ *refer* to round $\rho$.
- Prepare for round $\rho + 1$ *implies* Commit for round $\rho$.
- Primary proposes round $\rho + 1$ *after* it finished the prepare-phase for round $\rho$.

**Implies strict consecutive processing of rounds**

Overlapping *cannot* be combined with out-of-order processing!
The Single-Round Cost of PBFT with Overlapping

\[ \Delta_{\text{PBFT}} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta; \]

\[ T_{\text{PBFT}} = \frac{1}{\Delta_{\text{PBFT}}}; \]

Assumption: \( B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)

(\( \delta = 15 \text{ ms} \))

![Graph showing Throughput vs Number of Replicas](image1)

![Graph showing Throughput vs Message Delay](image2)
The Single-Round Cost of PBFT with Overlapping

\[ \Delta_{\text{PBFT}} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta; \]
\[ \Delta_{\text{op-PBFT}} = \frac{(n - 1)s_t + (n - 1)s_m}{B} + 2\delta; \]
\[ T_{\text{PBFT}} = \frac{1}{\Delta_{\text{PBFT}}}; \]
\[ T_{\text{op-PBFT}} = \frac{1}{\Delta_{\text{op-PBFT}}}. \]

Assumption: \( B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)

\( \delta = 15 \text{ ms} \)
The Single-Round Cost of PBFT with Overlapping

\[ \Delta_{\text{PBFT}} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta; \]

\[ \Delta_{\text{op-PBFT}} = \frac{(n - 1)s_t + (n - 1)s_m}{B} + 2\delta; \]

Assumption: \( B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)
The Single-Round Cost of PBFT with Overlapping

$$\Delta_{PBFT} = \frac{(n - 1)s_t + 2(n - 1)s_m}{B} + 3\delta;$$

$$\Delta_{op-PBFT} = \frac{(n - 1)s_t + (n - 1)s_m}{B} + 2\delta;$$

$$T_{PBFT} = \frac{1}{\Delta_{PBFT}};$$

$$T_{op-PBFT} = \frac{1}{\Delta_{op-PBFT}}.$$

Assumption: $B = 100 \text{ MiB/s}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
Implementation techniques for PBFT: Summary

**Batching** introduces very high round latencies.

**Out-of-order processing** has high implementation complexity.

**Overlapping** only provides limited gains.

**Assumption:** $n = 4$, $B = 100 \text{ MiB/s}$, $\delta = 15 \text{ ms}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
A PBFT-like design is at the basis of many consensus protocols.
A PBFT-like design is at the basis of many consensus protocols.

Technologies employed by PBFT-like consensus

- **Threshold signatures**: eliminate quadratic all-to-all communication.
- **Speculative execution**: execute before strong recovery guarantees are met.
- **Optimistic execution**: fully optimize for when the primary is correct.
- **Trusted components**: use hardware components that cannot behave Byzantine.

Here, we will only cover threshold signatures.
Almost All-to-All: \((n - 1)^2\) messages

All-to-All: \(n^2 - n\) messages

\(n^2 + (n - 1)^2 - n\) messages of constant size

Challenge: Reduce communication from \(O(n^2)\) to \(O(n)\) messages of constant size.
All-to-All Communication in PBFT

Almost All-to-All: $(n - 1)^2$ messages

Challenge: Reduce communication from $O(n^2)$ to $O(n)$ messages of constant size.
All-to-All Communication in PBFT

Almost All-to-All: \((n - 1)^2\) messages

All-to-All: \(n^2 - n\) messages

Challenge: Reduce communication from \(O(n^2)\) to \(O(n)\) messages of constant size.
All-to-All Communication in PBFT

\[ n^2 + (n - 1)^2 - n \] messages of constant size

Propose  Prepare  Commit
All-to-All Communication in PBFT

Almost All-to-All: \( (n-1)^2 \) messages
All-to-All: \( n^2 \) messages

\[ n^2 + (n - 1)^2 - n \] messages of constant size

Challenge: Reduce communication from \( \mathcal{O}(n^2) \) to \( \mathcal{O}(n) \) messages of constant size.
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

\[ n^2 - n \text{ messages} \]

Idea: All replicas send to one aggregator that then sends to all replicas.

Effectively reduced communication from \( O(n^2) \) to \( O(n(n - f)) \).
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

\[ n^2 - n \text{ messages} \]

\[ \Rightarrow \]

\[ (n - 1) \text{ messages} \]

\[ (n - 1) \text{ messages of constant size} \]

\[ (n - 1) \text{ messages of size } O(n - f) \]

Idea: All replicas send to one aggregator that then sends to all replicas.
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

\[ n^2 - n \text{ messages} \]

\[ \rightarrow \]

All-to-One:

\[ (n - 1) \text{ messages} \]

\[ (n - 1) \text{ messages of constant size} \]

\[ O(n - f) \text{ messages} \]

Idea: All replicas send to *one aggregator* that then sends to all replicas.

1. All replicas send their Commit messages to the aggregator.
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

\[ n^2 - n \text{ messages} \]

\[ \Rightarrow \]

\[ (n - 1) \text{ messages} \]

\[ (n - 1) \text{ messages of constant size} \]

\[ (n - 1) \text{ messages of size } O(n - f) \]

Idea: All replicas send to one aggregator that then sends to all replicas.

2. The aggregator combines \( n - f \) Commit messages into an aggregated message \( m_A \).
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

$\text{n}^2 - \text{n}$ messages

$\Rightarrow$

One-to-All:

$(\text{n} - 1)$ messages

Idea: All replicas send to one aggregator that then sends to all replicas.

3. The aggregator sends $m_A$ to all replicas.
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

\[ \sum_{n=0}^{n-1} m = \sum_{n=1}^{n-1} (n - 1) \text{ messages of constant size} \]

\[ \sum_{n=0}^{n-1} O(n - f) = \sum_{n=1}^{n-1} O(n - f) \text{ messages of size } O(n - f) \]

Idea: All replicas send to one aggregator that then sends to all replicas.

3. The aggregator sends \( m_A \) to all replicas of size \( O(n - f) \) each.
Tackling All-to-All via All-to-one-to-All Aggregation

Consider the commit phase

\[ n^2 - n \text{ messages} \]

\[ \rightarrow \]

\[ (n - 1) \text{ messages of constant size} \]

\[ (n - 1) \text{ messages of size } \mathcal{O}(n - f) \]

Idea: All replicas send to one aggregator that then sends to all replicas.

Effectively reduced communication from \( \mathcal{O}(n^2) \) to \( \mathcal{O}(n(n - f)) \).
Improving Aggregation with Threshold Signatures

Problem: An aggregated message of size $c$ will have size $\mathcal{O}(c(n - f))$.

- We have identical Commit messages from at-least $n - f$ replicas.
- Goal: aggregate these into a single message of size $\mathcal{O}(c)$ instead of $\mathcal{O}(c(n - f))$. 

- Crucially: we want to aggregate the digital signatures!

Solution: Using a $n : f$-threshold-signature scheme with public key $K$,

- Each replica has a unique private key.
- Replicas can produce partial signatures for value $v$ using their private key.
- Using $n - f$ partial signatures for $v$, one can produce a threshold signature.

Threshold signatures aggregate $n - f$ distinct signatures into a single constant-sized value.
Improving Aggregation with Threshold Signatures

Problem: An aggregated message of size $c$ will have size $\mathcal{O}(c(n - f))$.

▶ We have identical Commit messages from at-least $n - f$ replicas.
▶ Goal: aggregate these into a single message of size $\mathcal{O}(c)$ instead of $\mathcal{O}(c(n - f))$.
▶ Crucially: we want to aggregate the digital signatures!
Improving Aggregation with Threshold Signatures

Problem: An aggregated message of size $c$ will have size $O(c(n - f))$.

- We have identical Commit messages from at-least $n - f$ replicas.
- Goal: aggregate these into a single message of size $O(c)$ instead of $O(c(n - f))$.
- Crucially: we want to aggregate the digital signatures!

Solution: Using a $n : f$-threshold-signature scheme with public key $K$

- Each replica has a unique private key.
- Replicas can produce partial signatures for value $v$ using their private key.
- Using $n - f$ partial signatures for $v$, one can produce a threshold signature.
Improving Aggregation with Threshold Signatures

Problem: An aggregated message of size $c$ will have size $O(c(n - f))$.

- We have identical Commit messages from at-least $n - f$ replicas.
- Goal: aggregate these into a single message of size $O(c)$ instead of $O(c(n - f))$.
- Crucially: we want to aggregate the digital signatures!

Solution: Using a $n : f$-threshold-signature scheme with public key $K$

- Each replica has a unique private key.
- Replicas can produce partial signatures for value $v$ using their private key.
- Using $n - f$ partial signatures for $v$, one can produce a threshold signature.
Improving Aggregation with Threshold Signatures

Problem: An aggregated message of size $c$ will have size $O(c(n - f))$.

- We have identical Commit messages from at-least $n - f$ replicas.
- Goal: aggregate these into a single message of size $O(c)$ instead of $O(c(n - f))$.
- Crucially: we want to aggregate the digital signatures!

Solution: Using a $n : f$-threshold-signature scheme with public key $K$

- Each replica has a unique private key.
- Replicas can produce partial signatures for value $v$ using their private key.
- Using $n - f$ partial signatures for $v$, one can produce a threshold signature.

Threshold signatures aggregate $n - f$ distinct signatures into a single constant-sized value.
Consider the commit phase

\[ n^2 - n \text{ messages} \]

Effectively reduced communication from \( O(n^2) \) to \( O(n) \).

Similar change can be made to the prepare phase.
Consider the commit phase

\[ n^2 - n \text{ messages} \]

\[ \text{Commit} \]

\[ \Rightarrow \]

\[ \text{All-to-One:} \]

\[ (n - 1) \text{ partial signatures} \]
Consider the commit phase

\( n^2 - n \) messages

Commit

\( (n - 1) \) threshold signatures

Effectively reduced communication from \( O(n^2) \) to \( O(n) \).

Similar change can be made to the prepare phase.
All-to-one-to-All Aggregation with Threshold Signatures

Consider the commit phase

\[ n^2 - n \] messages

\( n - 1 \) partial signatures of constant size
\( n - 1 \) threshold signatures of constant size

Effectively reduced communication from \( O(n^2) \) to \( O(n) \).

Similar change can be made to the prepare phase.
All-to-one-to-All Aggregation with Threshold Signatures

Consider the commit phase

\[ n^2 - n \text{ messages} \]

Effectively reduced communication from \( O(n^2) \) to \( O(n) \).
Consider the commit phase

\[ n^2 - n \text{ messages} \]

Effectively reduced communication from \( O(n^2) \) to \( O(n) \).

Similar change can be made to the prepare phase.
Using Threshold Signatures in PBFT

- Both prepare and commit phase: from $2(n - 1)^2$ to $4(n - 1)$ messages.
- Consensus from *three* to *five* rounds: higher consensus and client latencies.
- High *computational cost* for the aggregator.
- Need recovery methods to deal with *faulty aggregators*.
Using Threshold Signatures in PBFT

- Both prepare and commit phase: from $2(n - 1)^2$ to $4(n - 1)$ messages.
- Consensus from three to five rounds: higher consensus and client latencies.
- High computational cost for the aggregator.
- Need recovery methods to deal with faulty aggregators.

Assumption: $B = 100$ MiB/s, $s_t = 4048$ B, $s_m = 256$ B

![Graph showing round duration and throughput vs message delay and number of replicas]
Limitations of Primary-Backup Consensus

- **Primary Send** \((n - 1)\) Propose, send \((n - 1)\) Commit.
- **Receive** \((n - 1)\) Prepare, receive \((n - 1)\) Commit.
- **Total:** \(m(n - 1) + 3(n - 1)\) ms.

- **Backup Send** \((n - 1)\) Propose, send \((n - 1)\) Commit.
- **Receive one** Propose, receive \((n - 2)\) Prepare, receive \((n - 1)\) Commit.
- **Total:** \(m + 4(n - 1)\) ms - ms.
Limitations of Primary-Backup Consensus

Primary Send \((n - 1)\) Propose, send \((n - 1)\) Commit.
Limitations of Primary-Backup Consensus

Primary Send \((n-1)\) Propose, send \((n-1)\) Commit.
Receive \((n-1)\) Prepare, receive \((n-1)\) Commit.
Limitations of Primary-Backup Consensus

Primary

Send \((n - 1)\) Propose, send \((n - 1)\) Commit.
Receive \((n - 1)\) Prepare, receive \((n - 1)\) Commit.
Total: \(m(n - 1)s_t + 3(n - 1)s_m\).
Limitations of Primary-Backup Consensus

Primary Send \((n - 1)\) Propose, send \((n - 1)\) Commit.
Receive \((n - 1)\) Prepare, receive \((n - 1)\) Commit.
Total: \(m(n - 1)s_t + 3(n - 1)s_m\).

Backup Send \((n - 1)\) Prepare, send \((n - 1)\) Commit.
Limitations of Primary-Backup Consensus

Primary  Send \((n - 1)\text{ Propose}, \text{ send } (n - 1)\text{ Commit.}\nReceive \((n - 1)\text{ Prepare}, \text{ receive } (n - 1)\text{ Commit.}\nTotal: \(m(n - 1)s_t + 3(n - 1)s_m.\)

Backup  Send \((n - 1)\text{ Prepare}, \text{ send } (n - 1)\text{ Commit.}\nReceive one Propose, receive \((n - 2)\text{ Prepare}, \text{ receive } (n - 1)\text{ Commit.}\)
Limitations of Primary-Backup Consensus

**Primary**  
Send \((n - 1)\) Propose, send \((n - 1)\) Commit.  
Receive \((n - 1)\) Prepare, receive \((n - 1)\) Commit.  
Total: \(m(n - 1)s_t + 3(n - 1)s_m\).

**Backup**  
Send \((n - 1)\) Prepare, send \((n - 1)\) Commit.  
Receive one Propose, receive \((n - 2)\) Prepare, receive \((n - 1)\) Commit.  
Total: \(ms_t + 4(n - 1)s_m - s_m\).
Limitations of Primary-Backup Consensus

Bandwidth ratio between primary and backups

\[ R_{PBFT-m} = \frac{m(n - 1)s_t + 3(n - 1)s_m}{m s_t + 4(n - 1)s_m - s_m}. \]
Limitations of Primary-Backup Consensus

Bandwidth ratio between primary and backups

\[ R_{PBFT-m} = \frac{m(n - 1)s_t + 3(n - 1)s_m}{ms_t + 4(n - 1)s_m - s_m}. \]

Assumption: \( s_t = 4048 \) B, \( s_m = 256 \) B
Maximum Throughput of Primary-Backup Consensus

Consider failure-free replication: Primary *only* proposes, no other communication.

\[ T_{\text{max}} = B \left( n - 1 \right) \text{s} \]

Assumption: \( B = 100 \text{ MiB/s} \), \( \text{s} = 4048 \text{ B} \), \( \text{sm} = 256 \text{ B} \)
Maximum Throughput of Primary-Backup Consensus

Consider failure-free replication: Primary *only* proposes, no other communication.

\[ T_{\text{max}} = \frac{B}{(n - 1)s_t}. \]
Maximum Throughput of Primary-Backup Consensus

Consider failure-free replication: Primary only proposes, no other communication.

\[ T_{\text{max}} = \frac{B}{(n - 1)s_t}. \]

Assumption: \( B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)
Maximum Throughput of Primary-Backup Consensus

Consider failure-free replication: Primary *only* proposes, no other communication.

\[ T_{\text{max}} = \frac{B}{(n - 1)s_t} \]

Assumption: \( B = 100 \text{ MiB/s} \), \( s_t = 4048 \text{ B} \), \( s_m = 256 \text{ B} \)
**Maximum Throughput of Primary-Backup Consensus**

Consider failure-free replication: Primary *only* proposes, no other communication.

\[ T_{\text{max}} = \frac{B}{(n - 1)s_t} \cdot \]

Assumption: \( B = 100 \text{ MiB/s} \), \( s_t = 4048 \text{ B} \), \( s_m = 256 \text{ B} \)
Maximum Throughput of Primary-Backup Consensus

Consider failure-free replication: Primary *only* proposes, no other communication.

\[ T_{\text{max}} = \frac{B}{(n - 1)s_t}. \]

Assumption: \( B = 100 \text{ MiB/s} \), \( s_t = 4048 \text{ B} \), \( s_m = 256 \text{ B} \)

<table>
<thead>
<tr>
<th>Number of Replicas</th>
<th>Throughput (round/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10^3</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>0.5</td>
</tr>
<tr>
<td>60</td>
<td>0.5</td>
</tr>
<tr>
<td>70</td>
<td>0.5</td>
</tr>
<tr>
<td>80</td>
<td>0.5</td>
</tr>
<tr>
<td>90</td>
<td>0.5</td>
</tr>
<tr>
<td>100</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Graph showing throughput as a function of the number of replicas.
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.

Consider the communication of one of the $z$ primaries.
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.

Consider the communication of one of the $z$ primaries.

- As *primary* of its own instance:
  - Send $(n - 1)$ Propose, send $(n - 1)$ Commit.
  - Receive $(n - 1)$ Prepare, receive $(n - 1)$ Commit.
  - Total: $m(n - 1)s_t + 3(n - 1)s_m$. 

- As *backup* of the other $z - 1$ instances ($z - 1$ times):
  - Send $(n - 1)$ Prepare, send $(n - 1)$ Commit.
  - Receive one Propose, receive $(n - 2)$ Prepare, receive $(n - 1)$ Commit.
  - Total: $(z - 1)(m \cdot s_t + 4(n - 1)s_m)$. 

Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

1 ≤ z ≤ n primaries: z simultaneous rounds of consensus that decide the next z requests.

Consider the communication of one of the z primaries.

► As primary of its own instance:
  Send (n − 1) Propose, send (n − 1) Commit.
  Receive (n − 1) Prepare, receive (n − 1) Commit.
  Total: \( m(n - 1)s_t + 3(n - 1)s_m \).

► As backup of the other \( z - 1 \) instances \((z - 1)\) times:
  Send (n − 1) Prepare, send (n − 1) Commit.
  Receive one Propose, receive (n − 2) Prepare, receive (n − 1) Commit.
  Total: \((z - 1)(ms_t + 4(n - 1)s_m - s_m)\).
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.

Consider the communication of one of the $z$ primaries.

- As *primary* of its own instance:
  
  Send $(n - 1)$ Propose, send $(n - 1)$ Commit.
  
  Receive $(n - 1)$ Prepare, receive $(n - 1)$ Commit.
  
  Total: $m(n - 1)s_t + 3(n - 1)s_m$.

- As *backup* of the other $z - 1$ instances ($(z - 1)$ times):
  
  Send $(n - 1)$ Prepare, send $(n - 1)$ Commit.
  
  Receive one Propose, receive $(n - 2)$ Prepare, receive $(n - 1)$ Commit.
  
  Total: $(z - 1)(ms_t + 4(n - 1)s_m - s_m)$.

$$T_{c-o00-PBFT}(z, m) = \frac{zmB}{(m(n - 1)s_t + 3(n - 1)s_m) + ((z - 1)(ms_t + 4(n - 1)s_m - s_m))}.$$
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.

$$T_{\text{c-ooo-PBFT-}}(z, m) = \frac{zmB}{(m(n - 1)s_t + 3(n - 1)s_m) + ((z - 1)(ms_t + 4(n - 1)s_m - sm))}.$$ 

Assumption: $B = 100 \text{ MiB/s}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

\[ 1 \leq z \leq n \text{ primaries: } z \text{ simultaneous rounds of consensus that decide the next } z \text{ requests.} \]

\[
T_{c-oo-PBFT-(z,m)} = \frac{zmB}{(m(n - 1)s_t + 3(n - 1)s_m) + ((z - 1)(ms_t + 4(n - 1)s_m - s_m))}.
\]

Assumption: \( B = 100 \text{ MiB/s}, s_t = 4048 \text{ B}, s_m = 256 \text{ B} \)
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

\[ 1 \leq z \leq n \] primaries: \( z \) simultaneous rounds of consensus that decide the next \( z \) requests.

\[
T_{c-ooo-PBFT-(z, m)} = \frac{zmB}{(m(n - 1)s_t + 3(n - 1)s_m) + ((z - 1)(ms_t + 4(n - 1)s_m - s_m))}.
\]

Assumption: \( B = 100 \text{ MiB/s} \), \( s_t = 4048 \text{ B} \), \( s_m = 256 \text{ B} \)
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.

$$T_{c-ooo-PBFT-(z, m)} = \frac{zmB}{(m(n-1)s_t + 3(n-1)s_m) + ((z-1)(ms_t + 4(n-1)s_m - s_m))}.$$  

Assumption: $B = 100 \text{ MiB/s}$, $s_t = 4048 \text{ B}$, $s_m = 256 \text{ B}$
Concurrent Consensus

Idea: Multiple instances of PBFT, each with a distinct primary

$1 \leq z \leq n$ primaries: $z$ simultaneous rounds of consensus that decide the next $z$ requests.

$$T_{c-ooo-PBFT-(z,m)} = \frac{zmB}{(m(n-1)s_t + 3(n-1)s_m) + ((z-1)(ms_t + 4(n-1)s_m - s_m))}.$$ 

Assumption: $B = 100$ MiB/s, $s_t = 4048$ B, $s_m = 256$ B