Efficient Fault-Tolerant Infrastructure for Cloud Computing

Xueyuan Su

Candidate for Ph.D. in Computer Science, Yale University
December 2013

Committee
Michael J. Fischer (advisor)
Dana Angluin
James Aspnes
Avi Silberschatz
Michael Mitzenmacher, Harvard (external reader)
1 Introduction

2 Assigning Tasks for Efficiency
   • An Idealized Model of Hadoop
   • Hadoop Task Assignment
   • Online Scheduling

3 Checkpointing with Unreliable Checkpoints
   • An Extended Model with Possible Checkpoint Failures
   • Results on Expected Job Completion Time
   • A Weighted Sum Theorem

4 Summary & Acknowledgements
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Data-intensive computing, big data.

- Store and process large volumes of data (search/transactions/scientific computing/etc...).
- Scalability, parallelism, and efficiency.

Solution: Cloud computing!
Cloud Computing

- Envisions shifting data storage and computing power from local servers to data centers.
- Includes both of
  - Applications delivered over the Internet,
  - Supporting hardware & software infrastructure.
Google’s **MapReduce** and Apache **Hadoop**.
- Built on large clusters of commodity machines.
- **Storage**: Data block replication in distributed file systems.
- **Execution**: Job partitioning into tasks for parallel execution.
Challenges for Infrastructure Design

- **Efficiency challenge**: Load balancing for fast job completion.
- **Data locality** is critical.
  - The time to load data *dominates* the actual computation time.
  - Running tasks on servers not holding input data → remotely load through network → communication cost plus disk access.
  - The more data needs to be remotely loaded → the more congested the network becomes → higher remote access cost.
**Fault tolerance challenge:** Inevitable failures due to scale.

**Completions of long-running jobs are affected by failures.**

- **Checkpointing & rollback.**
- Literature: Checkpoints are stored in **totally reliable storage.**
- Checkpoints **may fail** & their failure probability can be **significant.**
- **Faster** job completion can sometimes be achieved by using less **reliable but cheaper** checkpoints.
Efficiency: Assignment of tasks to servers for minimizing job completion time.

Fault tolerance: Checkpoint placements that consider possible checkpoint failure.
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Quick Overview of Hadoop Infrastructure

- **Storage**: Hadoop distributed file system (HDFS).
  - Files are split into large **equal-sized** blocks.
  - Each data block is **replicated** on multiple servers.
- **Job execution**: Break jobs down into small independent tasks.
  - Each task processes a **single** input data block.
  - Tasks are assigned to and run on **servers**.
  - Tasks assigned to the same server run **sequentially**.
  - Tasks assigned to different servers run **in parallel**.
  - Job completes when all tasks finish. Thus, the **job completion time** is the completion time of the **task that finishes last**.
**Challenge**: How to assign tasks to servers to simultaneously balance load and minimize cost of remote data access?
Components of the Model

- A set of **tasks** and a set of **data blocks**. Each data block has a corresponding task to be performed on it.
- A set of **servers** that store data blocks and perform tasks.
- A **data placement** that maps a data block to the set of servers on which **replicas** of that block are stored.
- A **task assignment** that specifies the server on which a task will run.
- A **cost function** that determines the run time of a task on a server to which it is assigned.
Every data block occurs on at least one server.

<table>
<thead>
<tr>
<th>Block</th>
<th>#Replicas</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>2</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
</tr>
</tbody>
</table>
Slide data blocks up out of the server rectangles and connect them by lines. This leads to a bipartite graph where each replica has an edge to its corresponding server.
Merge the replica nodes with the same block label together, giving the bipartite graph called the data placement graph $G_\rho$. 
Assigning Tasks to Servers

- A task is **local** if it is on the same server as its corresponding data block. Its cost is $w_{\text{loc}}$.
- A task is **remote** if it is on a different server from its corresponding data block. Its cost is $w_{\text{rem}}$.
- We assume $w_{\text{rem}} \geq w_{\text{loc}}$.

# remote tasks = 3

### Diagram

- **Local task**: Blue
- **Remote task**: Red

**Server 1**:
- Local: $a$, $d$
- Remote: $b$, $c$, $e$

**Server 2**:
- Local: $b$, $d$
- Remote: $a$, $e$

**Server 3**:
- Local: $a$, $c$
- Remote: $b$, $d$, $e$
Remote Cost

- Network becomes more congested with more remote tasks.
- Network congestion increases the cost for remote data access.
- To reflect the increased communication cost, $w_{\text{rem}}$ is a monotone increasing function of the number of remote tasks.
Reassigning task $c$ from server 3 to server 2 gives a worse result.

- The cost of task $c$ becomes remote cost.
- Remote costs for all remote tasks increase due to the increased number of remote tasks.

![Diagram showing reassignment of tasks and cost impact]

Server 1  Server 2  Server 3
# remote tasks = 3

Server 1  Server 2  Server 3
# remote tasks = 4
Server Load and Maximum Load

- The load $L_s^A$ of a server $s$ under assignment $A$ is the sum of costs introduced by all tasks that $A$ assigns to $s$.
- The maximum load $L^A$ of assignment $A$ is the maximum server load under $A$. In the following example, $L^A = L_2^A$. 

![Diagram showing server load and maximum load](image-url)
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Definition

The **HTA Optimization Problem** is, given a Hadoop system, to find a task assignment that **minimizes** the **maximum load**.

Definition

The **HTA Decision Problem** is, given a Hadoop system and a server capacity $k$, to decide whether there exists a task assignment with **maximum load $\leq k$**.
By reduction from 3SAT, we show that

**Theorem**

*The HTA decision problem is NP-complete.*

Actually the HTA problem remains hard even if the server capacity is 3, the cost function only has 2 values \(\{1, 3\}\) in its range, and each data block has at most 2 replicas.
Approximation Algorithms

We present and analyze two approximation algorithms.

- “Round Robin” algorithm computes assignments that deviate from the optimal by a factor \( \frac{w_{\text{rem}}}{w_{\text{loc}}} \).
- The flow-based algorithm “Max-Cover-Bal-Assign” computes assignments that are optimal to within an additive gap \( w_{\text{rem}} \).
Given a threshold $\tau$, the algorithm runs in two phases:

- **Max-cover** uses network flow to maximize the number of local tasks such that no single server is assigned more than $\tau$ local tasks.

- **Bal-assign** finishes the assignment by greedily assigning each unassigned task to a server with minimal load.
Example Data Placement

Server 1  Server 2  Server 3
Max-cover with Threshold $\tau = 4$

Max-cover assigns each server the maximum number of local tasks, subject to the threshold $\tau = 4$.
Bal-assign finishes the task assignment by greedily assigning each unassigned task to a server with minimal load.

Bal-assign finishes the task assignment by greedily assigning each unassigned task to a server with minimal load.
Optimal Assignment $O$

However, optimal assignment can do better...

\[ \tau = 4 \]

Server 1: $a$, $b$, $c$, $d$

Server 2: $i$, $j$, $k$

Server 3: $e$, $f$, $g$

$L^o$
But the difference is smaller than $w_{\text{rem}}$. 

![Diagram showing task assignment and scheduling]

Server 1: 
- $a$
- $b$
- $c$
- $d$

Server 2: 
- $g$
- $h$
- $i$
- $e$

Server 3: 
- $f$
- $j$
- $k$

$\tau = 4$ 

$L^A$ 

$\tau = 4$ 

$L^O$

31 Xueyuan Su

Efficient Fault-Tolerant Infrastructure for Cloud Computing
The Approximation Gap

By trying all possible thresholds $\tau$ and picking the best assignment, the flow-based algorithm performs surprisingly well.

**Theorem**

Let $n$ be the number of servers. For $n \geq 2$, the flow-based algorithm computes an assignment $A$ such that

$$L^A \leq L^O + \left(1 - \frac{1}{n-1}\right) w^O_{\text{rem}}$$
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Distinctions from the HTA problem:

- Tasks arrive **sequentially** in an online fashion.
- Tasks must be assigned to their **feasible** (local) servers.
- Tasks may have **different** costs (but do not depend on which server the tasks are assigned to).

**Goal:** Assign each task to a feasible server upon its arrival to minimize the maximum load.
The GREEDY algorithm:

- Chooses a feasible server with minimum load.
- Breaks ties arbitrarily.
The **GREEDY** algorithm:

- Chooses a feasible server with minimum load.
- Breaks ties arbitrarily.
The GREEDY algorithm:

- Chooses a feasible server with minimum load.
- Breaks ties arbitrarily.
Performance Measure and Our Approach

- Assume an optimal offline algorithm $\text{OPT}$ knows the task sequence in advance and has unbounded computational power.
- The competitive ratio is the maximum load ratio between GREEDY and OPT under all possible inputs.

**Theorem (Upper bound on competitive ratio of GREEDY)**

For any problem instance $\sigma$, let $n$ be the number of servers. Let $A_\sigma$ and $O_\sigma$ be computed by GREEDY and OPT, respectively.

$$\frac{L^{A_\sigma}}{L^{O_\sigma}} \leq \log n + 1 - \delta(\sigma),$$

where $0 \leq \delta(\sigma) \leq \log n$. $\delta(\sigma) = 0$ if and only if $O_\sigma$ perfectly balances load over all servers and $L^{A_\sigma}$ is a multiple of $L^{O_\sigma}$. 

Mathematical Structures for Competitive Analysis

The proof of the competitive ratio upper bound makes use of two novel structures.

- Witness graphs.
- Token Production Systems (TPSs).
Witness graphs are ordered weighted multidigraphs.

Each witness graph embeds two assignments and allow them to be compared in the same framework.
Witness Graph from Assignments

- Step 1: Add directions on edges.
- Step 2: Merge task vertices.
Witness Graph from Assignments (cont.)

- Step 3: Eliminate task vertices.
- Step 4: Merge server vertices.
- Step 5: Add edge weights and ordering.
Witness Graph from Assignments (cont.)

- Each server is represented by a vertex, and each task is represented by an edge.
- Server load under Assignment 1 equals incoming edge weight.
- Server load under Assignment 2 equals outgoing edge weight.
- The total task load equals the total edge weight.
A token production system consists of a list of **buckets** and some number of **tokens**.

A set of vectors called **actions** rule the production and consumption of tokens in buckets. Each action has a **cost**.

A sequence of actions form an **activity** and its cost is the sum of action costs.
A witness graph can be embedded into a token production system.

Each vertex is represented by an action.

The maximum weight of incoming edges over all vertices equals the number of buckets.

The total edge weight equals the cost of an activity.
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Hardware failures are the norm, not the exception, in large systems.

The chance of job completion without any failures becomes exponentially small with increasing job processing time.

Assume a job execution fails. After faulty parts are fixed or replaced, jobs must be restarted and lost data must be recomputed, incurring substantial overhead.
Checkpoints Avoid Restart Overhead

- **Checkpoint**: Save the job state from time to time.
- **Rollback**: When a failure is detected, a rollback operation restores the job state to one previously saved in a checkpoint, and recomputation begins from there.
- This avoids always restarting the job from the beginning.
However, Checkpoints Might Also Fail

- Checkpoints **may fail** and their failure probability can be significant.
- Faster job completion can sometimes be achieved by using less reliable but cheaper checkpoints.
Impact of Checkpoint Failures

Challenges:

- How to evaluate the impact of checkpoint failures on job completion time?
- How to place checkpoints to minimize the expected job completion time.
Job Execution with Unreliable Checkpoints

- A set of tasks $t_1, t_2, \ldots, t_n$ to be executed sequentially.
- Each run of $t_i$ succeeds with positive probability $p_i$.
- Upon success, store the output of $t_i$ into checkpoint $c_{i+1}$.
- When a run of $t_i$ fails, recovery from checkpoint $c_i$ is attempted. With probability $q_i$, the recovery succeeds and the job is resumed from there. If the access to $c_i$ fails, access to the previous checkpoint $c_{i-1}$ is attempted and repeat.
- Restarting the job from the beginning is always possible.
When a task fails, accesses to checkpoints are attempted.

When both checkpoints fail, the job is restarted from the beginning.
When another task fails, accesses to checkpoints are attempted.

In this example, the access to the most recent checkpoint fails, but the second most recent one succeeds. Recovery resumes from there.
Finally the job successfully completes.
Two Placement Models

- The **discrete (modular program)** model:
  - Task lengths are **pre-determined**.
  - Checkpoints can only be placed at task **boundaries**.
  - Task success probability may be independent of task length.
  - A task failure is detected at the **end** of the task run.

- The **continuous** model:
  - Checkpoints can be placed at **arbitrary** points in the job.
  - Task lengths are determined by distance between adjacent checkpoints.
  - Task failures follow the **Poisson** distribution. Task success probability depends on task length and failure rate.
  - A task failure is detected **immediately** after its occurrence.
Job Execution as a Markov Chain

- For each $1 \leq i \leq n$, state $t_i$ represents the start of task $t_i$, and state $c_i$ represents the attempted access to checkpoint $c_i$.
- For convenience, state $t_{n+1}$ represents the job completion.
- A job execution is a random walk from state $t_1$ to state $t_{n+1}$.
Outline

1 Introduction

2 Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3 Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4 Summary & Acknowledgements
Let $S_i$ be the number of successful runs of task $t_i$ during a job execution.

The sample space consists of all possible job executions.

Then the expectation of $S_i$ satisfies the recurrence relation

$$
E[S_n] = 1
$$

$$
E[S_{i-1}] = \left(\frac{1 - q_i}{p_i}\right) E[S_i] + q_i, \text{ for all } i \neq 1
$$
Extreme values of $q$:

$$E[S_{i-1}] = \left(\frac{1 - q_i}{p_i}\right) E[S_i] + q_i$$

- If $q_i = 0$, $E[S_{i-1}] = E[S_i]/p_i$ (exponential growth). This is the case without any checkpoints.
- If $q_i = 1$, $E[S_{i-1}] = 1$ (constant). This is the case with totally reliable checkpoints.
Consider the uniform case where $p_i = p$ and $q_i = q$ for all $i$.

- $p + q$ is critical in determining the growth rate of expected job completion time as a function of job length $n$.
- Let $r = (1 - q)/p$.

\[
T = \begin{cases} 
\frac{q}{p(1-r)} n + \Theta(1) = \Theta(n) & \text{if } p + q > 1, \\
\frac{q}{2p} n^2 + \frac{2-q}{2p} n = \Theta(n^2) & \text{if } p + q = 1, \\
\frac{1}{p(r-1)} r^n + \frac{q}{p(1-r)} n + \Theta(1) = \Theta(r^n) & \text{if } p + q < 1.
\end{cases}
\]

- System designers should configure checkpoints such that $p + q > 1$. 
Sometimes the storage for checkpoints is limited. Assume at most $k$ checkpoints can be stored at the same time.

Consider the uniform case where $p_i = p$ and $q_i = q$ for all $i$ such that $p + q > 1$.

A simple retention policy keeps only the most recent $k$ checkpoints and discards older ones.

When $k = \Omega(\log n)$, the expected job completion time is $T = \Theta(n)$. The penalty is only a constant factor.
In many systems, it is reasonable to make checkpoints over regular intervals, e.g., once an hour. One might want to decide the optimum checkpoint interval to minimize expected job completion time.

Let $q$ be checkpoint reliability, $g$ be the cost for generating a checkpoint, and $M$ be mean time between failure.

When $g \ll M$, a first order approximation to the optimum checkpoint interval is

$$w = \sqrt{\left(\frac{q}{2 - q}\right) 2gM}$$

This expression generalizes the result $w = \sqrt{2gM}$ in the literature for totally reliable checkpoints.
Two Symmetry Properties

- Faster expected job completion can be achieved with the freedom to place checkpoints over non-equidistant points.
- Represent task lengths by a vector $\mathbf{w} = (w_1, \cdots, w_n)$. Let $\mathbf{w}^R = (w_n, \cdots, w_1)$ be its reversal.
- Two symmetry properties:
  \[
  T(\mathbf{w}) = T(\mathbf{w}^R) \\
  T(\mathbf{w}) \geq T\left(\frac{\mathbf{w} + \mathbf{w}^R}{2}\right)
  \]
- Both symmetry properties are proved by induction and multiple applications of a weighted sum theorem.
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements

Efficient Fault-Tolerant Infrastructure for Cloud Computing
Concatenation of Jobs

Concatenating two jobs with a checkpoint results in a larger job.
Theorem (Weighted Sum)

Let $T(q)$ be the expected completion time of the concatenation of two jobs with a checkpoint of reliability $q$. $T(q)$ satisfies

\[ T(q) = qT(1) + (1 - q)T(0) \]

In other words, $T(q)$ is the weighted sum of the two extreme cases where checkpoints are either totally reliable or totally unreliable.
Proof of the Weighted Sum Theorem

For job $i$, define

- $W_i$: expected completion time (left in right out).
- $W_i^R$: expected reverse completion time (right in right out).
- $p_i$: success probability.
Proof of the Weighted Sum Theorem (cont.)

\[ T(q) = W_1 + W_2 + (1 - q) \left( \frac{1}{p_2} - 1 \right) W_1^R \]

- **W_1 + W_2**: Both jobs must complete.
- **(1 − q)(1/p_2 − 1)**: On average, job 2 fails for \((1/p_2 − 1)\) times. Each time job 2 fails, access to the connecting checkpoint is attempted. Each access fails with probability \((1 − q)\).
- **W_1^R**: Each time when the connecting checkpoint fails, job 1 must reverse complete.
Proof of the Weighted Sum Theorem (cont.)

\[ T(q) = W_1 + W_2 + (1 - q) \left( \frac{1}{p_2} - 1 \right) W_1^R \]

Plugging \( q = 1 \) and \( q = 0 \) into above equation gives

\[
q T(1) + (1 - q) T(0)
= q (W_1 + W_2) + (1 - q) \left( W_1 + W_2 \left( \frac{1}{p_2} - 1 \right) W_1^R \right)
= W_1 + W_2 + (1 - q) \left( \frac{1}{p_2} - 1 \right) W_1^R
= T(q)
\]
Outline

1. Introduction

2. Assigning Tasks for Efficiency
   - An Idealized Model of Hadoop
   - Hadoop Task Assignment
   - Online Scheduling

3. Checkpointing with Unreliable Checkpoints
   - An Extended Model with Possible Checkpoint Failures
   - Results on Expected Job Completion Time
   - A Weighted Sum Theorem

4. Summary & Acknowledgements
Efficiency

- A model for studying the problem of assigning tasks to servers so as to minimize job completion time.
- \(\mathcal{NP}\)-completeness result for HTA problem.
- Good approximation algorithms for task assignment.
- New competitive ratio bound for online GREEDY.
A model that includes checkpoint reliability in evaluating job completion time.

\[ p + q \] determines the growth rate of expected completion time.

Logarithmic checkpoint retention with constant factor penalty.

First order approximation to optimum checkpoint interval.

Two symmetry properties for non-equidistant placements.
Sidenotes on Non-Theoretical Work

- **Google**: Chunk Allocation Algorithms for Colossus (the 2nd gen. of Google File System). *Do No Evil. No publication!*
- **ORACLE**: Oracle In-Database Hadoop: When MapReduce Meets RDBMS. *In SIGMOD 2012 & a feature in Oracle Database 12c.*
Before Finishing My Talk...

Acknowledgements
Hopeless Little X.

P vs. NP?
The Wise Mike

- Optimism + Persistence + Preciseness
- P vs. NP?
Inspiring Committee & Friends

- Theory?
- Optimism
  + Persistence
  + Preciseness

- P vs. NP?

Image: Diagram illustrating the theory and its components.
Supportive Family

- Theory?
- Optimism + Persistence + Preciseness

- P vs. NP?
Theory?
Optimism + Persistence + Preciseness

Thank You!
Outline

5 Proof Structure for HTA NP-Completeness

6 A Weaker Approximation Bound

7 Recurrence Relation for Successful Task Runs

8 Proof of the First Symmetry

9 Proof Sketch of the Second Symmetry
Proof Method

Given a 3CNF formula, we construct an instance of the HTA decision problem such that

- the local cost is 1,
- the remote cost is 3,
- the server capacity is 3.

The resulting HTA instance has a task assignment of maximum load at most 3 if and only if the 3CNF formula is satisfiable.

Note that a server can be assigned at most 3 local tasks or 1 remote task.
Let $C_u = (l_{u1} \lor l_{u2} \lor l_{u3})$ be a clause in the 3CNF formula and each $l_{uv}$ be a literal.

Construct a corresponding clause gadget, containing 1 clause server $C_u$, 3 literal tasks $l_{u1}, l_{u2}, l_{u3}$ and 1 auxiliary task $a_u$.

The server can only accept 3 local tasks, and thus 1 of the tasks needs to be assigned elsewhere.
There exists a feasible task assignment such that:

- The **auxiliary task** is assigned locally to its corresponding clause server.
- All **false literal tasks** are assigned locally to their corresponding clause servers.

Therefore, some literal task must be assigned elsewhere. This will imply that the corresponding literal of the 3CNF formula is **true**.

Please refer to my thesis for more details...
Outline

5. Proof Structure for HTA NP-Completeness
6. A Weaker Approximation Bound
7. Recurrence Relation for Successful Task Runs
8. Proof of the First Symmetry
9. Proof Sketch of the Second Symmetry
Lemma

Let $n$ be the number of servers, $L^A \leq L^O + (1 - 1/n) w_{rem}^O$.

Proof intuition:
- Max-cover guarantees no fewer local tasks than $O$.
- Bal-assign balances remaining remote tasks.
Proof of the Approximation Bound

- Let $K^A$ and $K^O$ be the total number of local tasks under assignments $A$ and $O$, respectively.
- Max-cover uses network flow to maximize the number of local tasks with respect to the threshold $\tau$.
- When $\tau^A = \tau^O$, we have $K^A \geq K^O$.

![Diagram showing task distribution across servers with different thresholds.]
The number of local tasks $K^A \geq K^O$ implies that $A$ assigns no more remote tasks than $O$.

By the **monotonicity** of $w_{\text{rem}}$, we have $w^{A}_{\text{rem}} \leq w^{O}_{\text{rem}}$.

Let $H^A$ and $H^O$ be the total load under assignments $A$ and $O$, respectively. It is straightforward that $H^A \leq H^O$. 
The maximum load cannot be smaller than the average. Thus,

\[ L^O \geq H^O / n \]

This implies that

\[ H^O \leq n \cdot L^O \]
Proof of the Approximation Bound (cont.)

Consider the server with maximum load (server 2) and the last task assigned to it (task \( g \)).

- If \( g \) is assigned by max-cover, then \( L^A = L^O = \tau \cdot w_{loc} \).
- If \( g \) is assigned by the greedy bal-assign, before \( g \) is assigned,
  \[ \forall s \neq 2, L_s^A \geq L^A - w_{rem}^A \]
  \[ \rightarrow H^A \geq (n - 1) \cdot (L^A - w_{rem}^A) + L^A \]
Proof of the Approximation Bound (cont.)

Combining the four inequalities completes the proof.

\[
\begin{align*}
    w_{rem}^A & \leq w_{rem}^O \\
    H^A & \leq H^O \\
    H^O & \leq n \cdot L^O \\
    H^A & \geq (n - 1) \cdot (L^A - w_{rem}^A) + L^A \\
    \rightarrow & \quad L^A \leq L^O + (1 - 1/n) w_{rem}^O
\end{align*}
\]
Outline

5. Proof Structure for HTA NP-Completeness
6. A Weaker Approximation Bound
7. Recurrence Relation for Successful Task Runs
8. Proof of the First Symmetry
9. Proof Sketch of the Second Symmetry
We prove this theorem by counting arguments.

- \( \mathbb{E}[S_n] = 1 \) as the job completes once task \( t_n \) succeeds.
- For \( 2 \leq i \leq n + 1 \), consider task \( t_{i-1} \).
  - \( t_{i-1} \) succeeds at least once.
  - Each time when checkpoint \( c_i \) fails, task \( t_{i-1} \) needs to be successfully re-executed to restore its output.
  - Let \( D_i \) be the number of times that checkpoint \( c_i \) fails.

\[
\mathbb{E}[S_{i-1}] = 1 + \mathbb{E}[D_i]
\]
Consider checkpoint $c_i$.

- Each time when task $t_i$ fails or checkpoint $c_{i+1}$ fails, an attempted access to checkpoint $c_i$ is made.
- Each attempted access fails with probability $(1 - q_i)$.
- Let $F_i$ be the number of times task $t_i$ fails.

$$
E[D_i] = (1 - q_i)(E[F_i] + E[D_{i+1}])
$$
Failed Task Runs vs. Successful Task Runs

Task runs are repeated Bernoulli trials, and thus

\[ E[F_i] = \left( \frac{1}{p_i} - 1 \right) E[S_i] \]
Recurrence of Successful Task Runs

Combining the three equalities gives

\[ E[S_{i-1}] = 1 + E[D_i] \]
\[ = 1 + (1 - q_i)(E[F_i] + E[D_{i+1}]) \]
\[ = 1 + (1 - q_i) \left( \left( \frac{1}{p_i} - 1 \right) E[S_i] + E[S_i] - 1 \right) \]
\[ = \frac{1 - q_i}{p_i} E[S_i] + q_i \]
Outline

5 Proof Structure for HTA NP-Completeness
6 A Weaker Approximation Bound
7 Recurrence Relation for Successful Task Runs
8 Proof of the First Symmetry
9 Proof Sketch of the Second Symmetry
Proof of the First Symmetry

Inductive proof:

• The base case with \( \ell = 1 \) task is trivially true.
• Assume the claim holds for \( \ell = n - 1 \) tasks.
• Now consider the inductive step with \( \ell = n \) tasks.
Apply the weighted sum theorem to the checkpoint after the first task in \((w_1, \cdots, w_n)\),

\[
T_n(w_1, \cdots, w_n) = q [T_1(w_1) + T_{n-1}(w_2, \cdots, w_n)] + (1 - q) T_{n-1}(w_1 + w_2, w_3, \cdots, w_n)
\]
Applying the weighted sum theorem to the checkpoint before the last task in \((w_n, \cdots, w_1)\),

\[
T_n(w_n, \cdots, w_1) = q \left[ T_{n-1}(w_n, \cdots, w_2) + T_1(w_1) \right] \\
+ (1 - q) T_{n-1}(w_n, \cdots, w_3, w_2 + w_1)
\]
By induction hypothesis,

\[ T_{n-1}(w_2, \cdots, w_n) = T_{n-1}(w_n, \cdots, w_2) \]

\[ T_{n-1}(w_1 + w_2, w_3, \cdots, w_n) = T_{n-1}(w_n, \cdots, w_3, w_2 + w_1) \]
Combining the four equalities completes the proof.

\[
T_n(w_1, \cdots, w_n) = q \left[ T_1(w_1) + T_{n-1}(w_2, \cdots, w_n) \right] \\
+ (1 - q) T_{n-1}(w_1 + w_2, w_3, \cdots, w_n) \\
= q \left[ T_1(w_1) + T_{n-1}(w_n, \cdots, w_2) \right] \\
+ (1 - q) T_{n-1}(w_n, \cdots, w_3, w_2 + w_1) \\
= T_n(w_n, \cdots, w_1)
\]
Outline

1. Proof Structure for HTA NP-Completeness
2. A Weaker Approximation Bound
3. Recurrence Relation for Successful Task Runs
4. Proof of the First Symmetry
5. Proof Sketch of the Second Symmetry
Consider length vector \( \mathbf{w} = (w_1, \cdots, w_n) \), where \( n \) is the vector length. For any \( x, y \geq 0 \), define a family of functions \( f_n \).

- \( f_0(\mathbf{w}, x, y) = 0 \).
- For \( n \geq 1 \),

\[
f_n(\mathbf{w}, x, y) = T_n(x + w_1, w_2, \cdots, w_n) + T_n(w_1, \cdots, w_{n-1}, w_n + y)
\]
Property of Helper Functions

Lemma

Let \( w = (w_1, \cdots, w_n) \) be a length vector and \( w^R = (w_n, \cdots, w_1) \) be its reversal. For any \( n \geq 0 \) and any \( x, y \geq 0 \), we have

\[
    f_n(w, x, y) \geq \frac{1}{2} f_n\left( \frac{w + w^R}{2}, \frac{x + y}{2}, \frac{x + y}{2} \right)
\]

This Lemma is proved by induction on \( n \) and multiple applications of the weighed sum theorem. Then it follows immediately that

\[
    T(w) = \frac{1}{2} f(w, 0, 0) \geq \frac{1}{2} f\left( \frac{w + w^R}{2}, 0, 0 \right) = T\left( \frac{w + w^R}{2} \right)
\]