The Thesis Structure

1. Introduction

+ Overview
+ Problem Formulation & Motivation
+ Research Questions
+ Methodology & Contributions

RQ1: How can we model the behavior of reactive load balancing to understand the limit in distributed memory systems?

RQ2: How can we proactively balance the load of task parallel applications at runtime?

SQ1: How can we predict the load of tasks to support proactive load balancing at runtime?

SQ2: How can we proactively co-schedule tasks to balance the load?

2. From Work Stealing to Reactive Load Balancing

+ **Preliminaries**

+ **Instrument**: Task-based Programming Models

Distributed Memory Systems

+ **Object**: Task Parallel Apps

+ **Constraints**: *Performance Slowdown* | *Communication Overhead*

+ **Target**: *Load Balancing (Speedup, Makespan)*

+ **Related Work**

3. Performance Modeling

+ Input:

T, P	w	R_{imb}	d	O_b
task distribution	task runtime	imbalance ratio	comm. delay	balancing overhead

+ Model: $F(T, P, w, R_{imb}, d, O_b)$

4. Proactive Load Balancing (Proactive LB)

A proposed scheme for proactive LB approach

Main Execution Threads executing tasks | executing tasks

Dedicated Assistant Thread modularized by components

<monitoring> <characterization> <load_prediction>

Three strategies for proactive LB

- Feedback Task Offloading
- ML-based Task Offloading
  - Stage 1: Task characterization
  - Stage 2: Online load prediction
  - Stage 3: Proactive Task Offloading
- Co-scheduling Tasks across Multiple Applications

5. Proof of Concepts

+ Simulator: *to evaluate the model*

+ **C++** Implementation: plugin tool upon *Chameleon*

+ **Python** for visualizing and checking prediction models

6. Evaluation

+ Synthetic microbenchmarks: MxM, Nbody, ...

+ Real use case: Sam(oa)^2

7.Conclusions & Future Work

+ Conclusion

+ Extended from the direction of performance modeling

+ Extended from the direction of proactive load balancing