ポストペタスケール計算機環境に向けた高可用分散協調セルフスケジューリング機構の提案

竹房あつ子，中田秀基，池上努，田中良夫
産業技術総合研究所 情報技術研究部門

ポストペタスケール計算機環境

- エクサスケール計算機が2018年に実現予定
  [IESP-roadmap-1.1, hpci-roadmap]
  - 十万プロセッサ，数千万コア規模
  - 故障発生間隔（MTBF）は1日～5分になると予測

  →故障が発生しても計算を継続実行できる耐障害性が必要

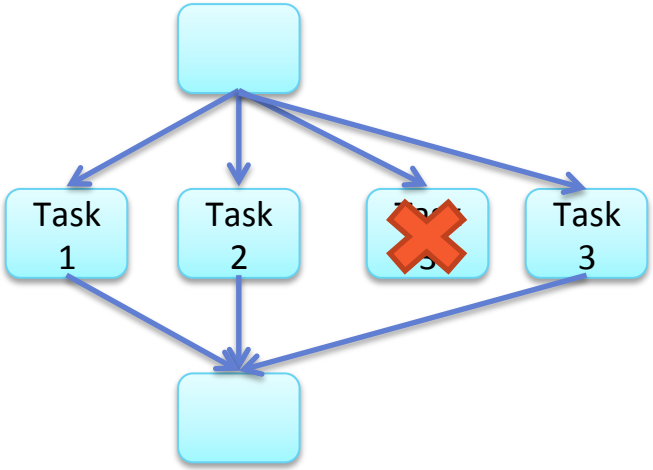
- ポストペタスケール計算機のプログラミングモデル
  - 従来のデータ並列型アプリケーションでは，強スケーリングは困難

  →階層型タスク並列処理が有望［FOXプロジェクト］
    - タスク並列の各タスクにデータ並列が内包

タスク並列処理の耐障害設計

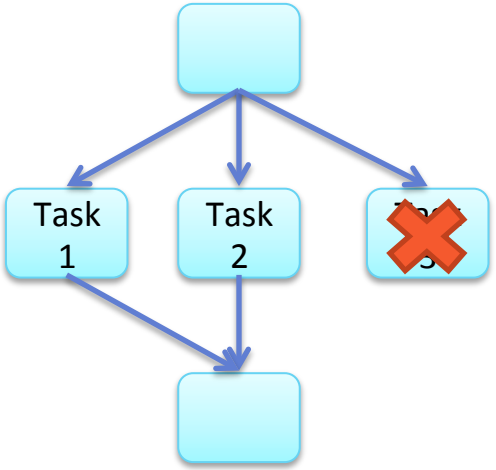
- 失敗箇所を再計算して実行フローを維持する方法

  E.g., フラグメント分子軌道法（FMO）



- アルゴリズムそのものに冗長性を内包させる方法

  E.g., ブロック櫻井・杉浦法



計算ロジックと無関係な部分で煩雑なコーディングが必要

耐障害アプリケーションフレームワーク

- "耐障害アプリケーションフレームワーク"を提案
- 耐障害性を備えるスケーラブルな階層型タスク並列アプリケーションの開発を支援
- アプリケーションを継続実行させる仕組みとして，データストア機構，資源管理機構を提供
  - データストア機構：→高信頼データストア機構を別途開発
    計算に必要となるデータを障害から保全
  - 資源管理機構：
    故障箇所を避けながら適切な計算ノードで計算を実行

資源管理機構の課題

- スケーラビリティ
  - 計算ノード数の増加やタスクの粒度により，資源管理へのリクエスト数が増大
  - スケーラブルな資源管理機構が必要
- 資源管理機構そのものの耐障害性
  - 資源管理機構自体に対する耐障害性は不十分
  - 障害が発生しても，継続した資源管理処理が不可欠
- 資源管理情報の永続化
  - 継続処理には，資源管理で扱う情報の永続化が必要
    - タスクID，各タスクの処理状況，計算ノードの情報
  - ファイルシステムは遅延大，分散KVSでは一貫性が課題

本研究の成果

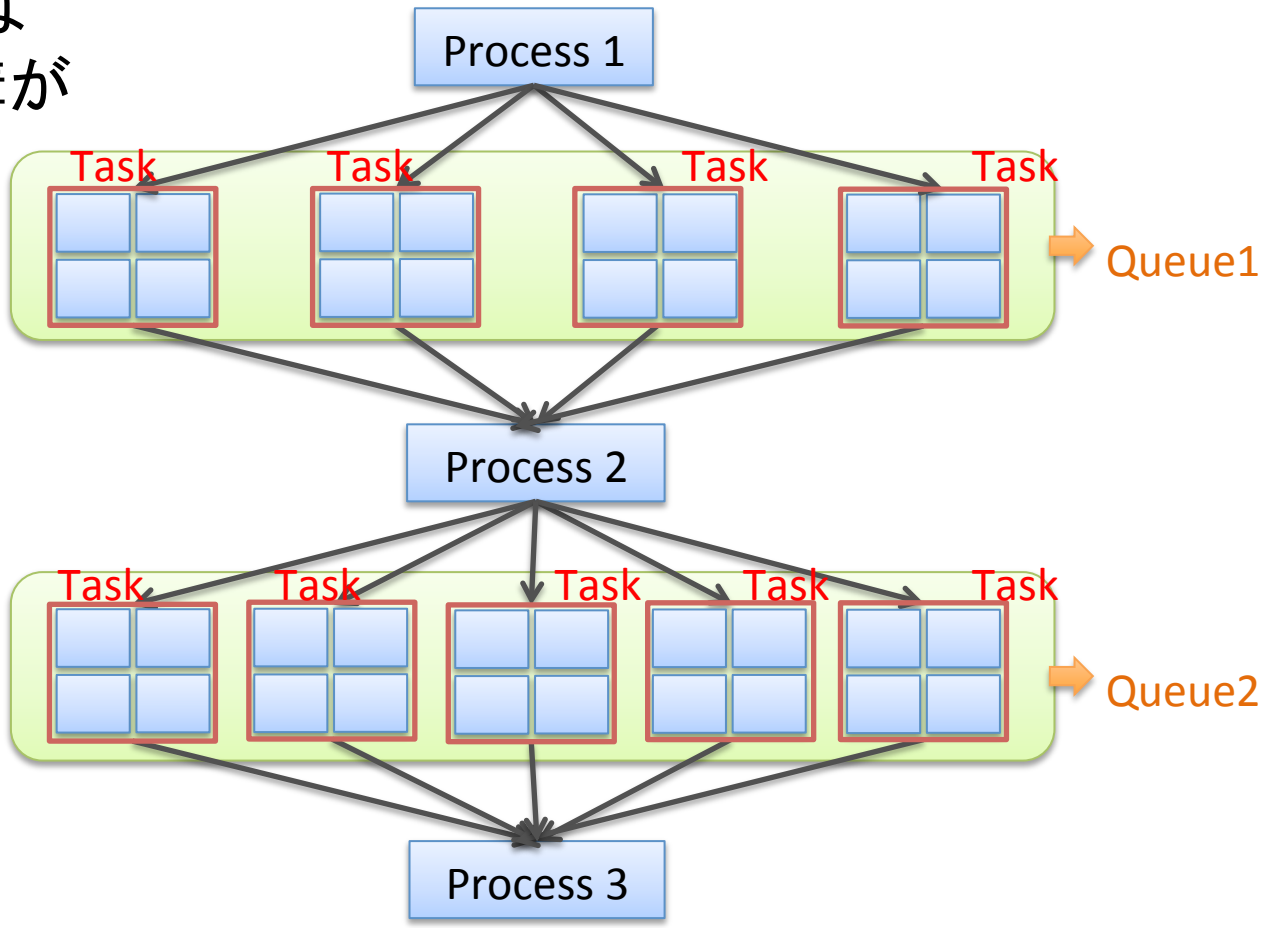
- スケーラブルかつ可用性の高い分散協調セルフスケジューリング機構を提案
  - 複数資源管理プロセスを分散協調させ，タスク群をキューで管理
  - 各計算ノードのデーモンプロセスが自律的にタスクを実行
  - 計算ノードの死活監視を行い，故障発生時にタスクを再実行／削除する仕組みを提供
  - 資源管理プロセスの耐障害性と資源管理情報の永続化
- Apache ZooKeeperを用いた試験実装
  - 故障が発生しても継続して実行できることを確認

発表概要

- 対象アプリケーションとプログラムイメージ
- 高可用分散協調セルフスケジューリング機構の設計
- Apache ZooKeeperを用いた試験実装
- 関連研究
- まとめと今後の課題

対象アプリケーション

- 階層型タスク並列アプリケーション
  - タスク(Task)は資源管理機構が資源割り当てを行う単位
  - 各タスクは数十～数百コア並列
  - タスク単位で再実行／削除が指定可能



プログラムイメージ

```
FunctionId FM01;            // タスクが実行する関数のID
int nTasks, nCores;         // 総タスク数，タスクの実行に必要なコア数

TaskQueue queue = new TaskQueue(FM01, nCores);  //タスクキューの生成
for (int i = 0; i < nTasks, i++) {
  queue.submit(IN[i], OUT[i], RESUBMIT_TRUE);  //キューにタスクを投入
                    // IN，OUTは入出力パラメータのキー配列
                    // RESUBMIT_TRUEで失敗時の再実行／削除を指定
}
queue.waitAll();   // 全てのタスクの終了待ち
```

関数のコードの実体，各入出力パラメータの値の管理は，
アプリケーションフレームワークのデータストア機構を利用

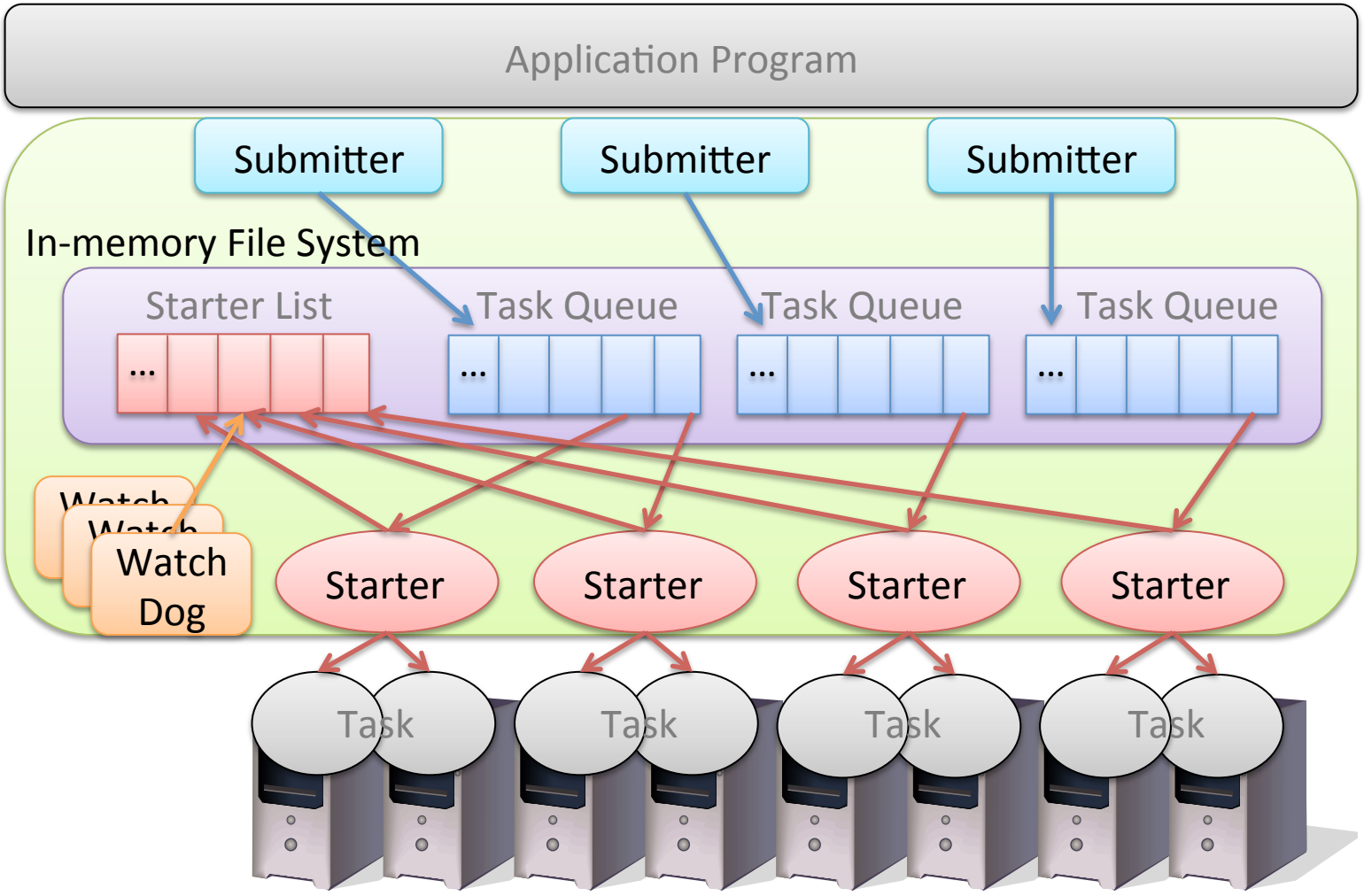
発表概要

- 対象アプリケーションとプログラムイメージ
- 高可用分散協調セルフスケジューリング機構の設計
- Apache ZooKeeperを用いた試験実装
- 関連研究
- まとめと今後の課題

高可用分散協調セルフスケジューリング機構（スケジューリング機構）の機能

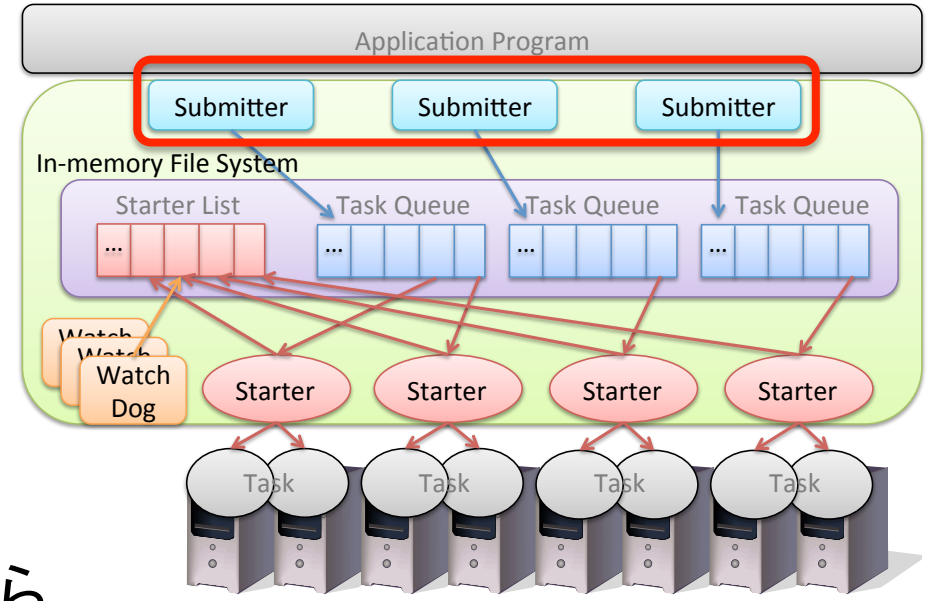
1. タスクキューへのタスク投入 → Submitter
2. タスクキュー内のタスクの実行 → Starter
3. 計算ノードの死活監視 → WatchDog
4. 障害発生時のタスク再実行／削除
5. 資源管理情報のスケーラブルな管理と永続化 → In-memory FS

スケジューリング機構のシステムアーキテクチャ



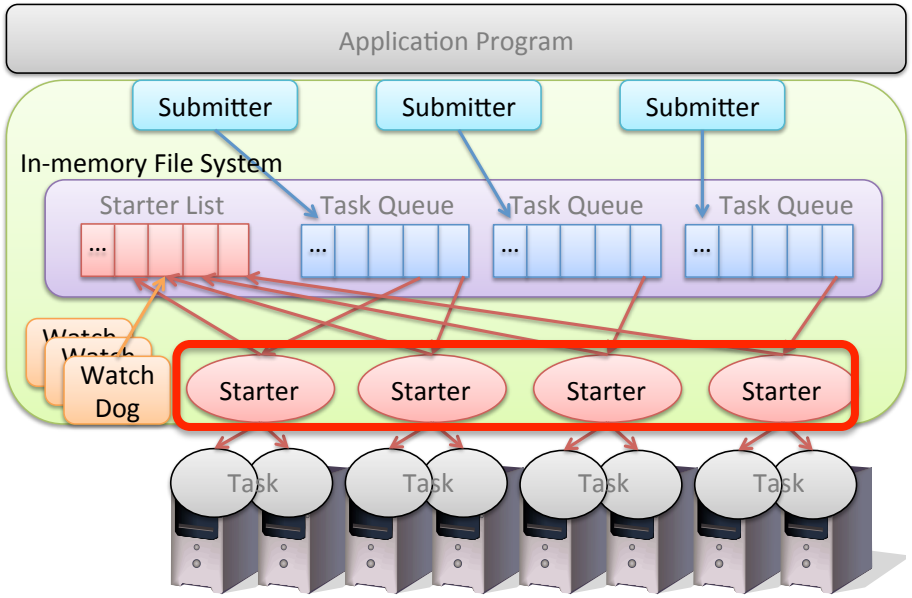
Submitter：タスク投入，再実行／削除

- タスクをタスクキューに投入
  - タスクキューではタスクのステータス情報を管理
  - ステータス：INITIAL, RUNNING, COMPLETED, FAILED
- タスクの終了を確認したら，データストアから結果を取得
- タスク実行の失敗を検知したら，再実行→キューに再投入／削除→削除処理



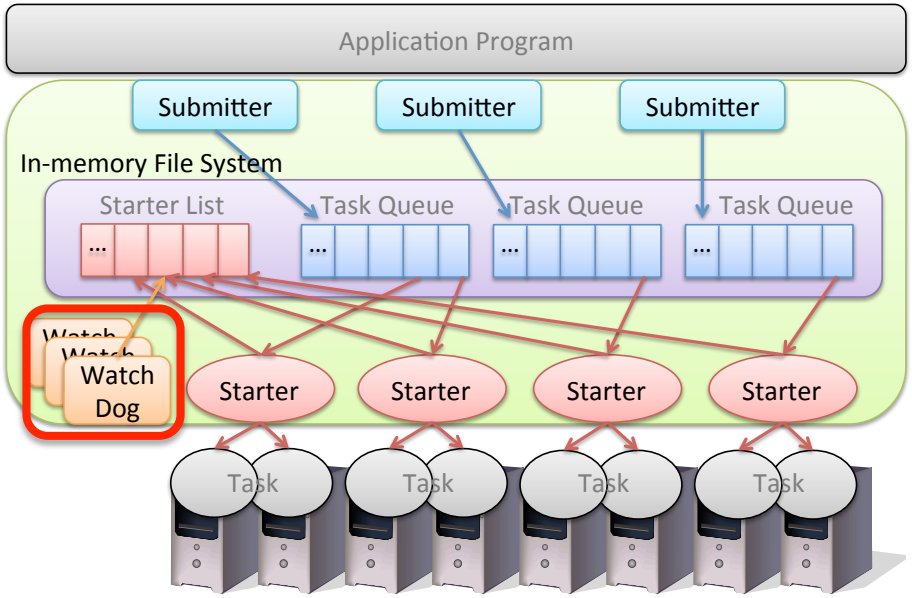
Starter：タスクの実行

- 各Starterは自律的にタスクキューの先頭タスクを取り出し，計算ノード上で実行
- 実行結果はデータストアに格納
- Starterは自身の情報をStarterリストに登録
  →死活監視に利用
- タスクの終了後／故障発生による再起動後は，新たなタスクを取得・実行へ



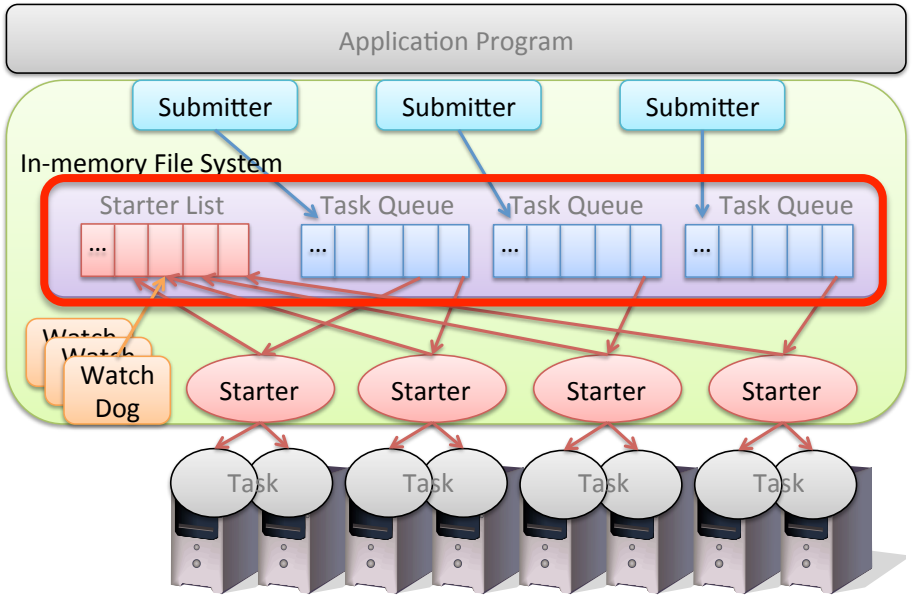
WatchDog：死活監視

- 定期的にタスクキューとStarterリストを監視
- RUNNING状態かつStarterに故障がある場合，タスクをFAILED状態へ
  →Submitterが再実行／削除
- WatchDogの耐障害性のため，複数WatchDogを立ち上げ，リーダーが処理



In-memory FS：資源管理情報管理，永続化

- ラック単位で複数プロセスを立ち上げ，分散協調して簡易ファイルシステムを構成
  - データ複製を共有
  - 各プロセスで永続化
  - 故障後も，プロセスグループに参加することで再構成可能
- データ共有の負荷軽減のため，小さいデータのみ扱う
  - 大きなデータはキー情報のみ格納
  - データそのものはデータストアに格納

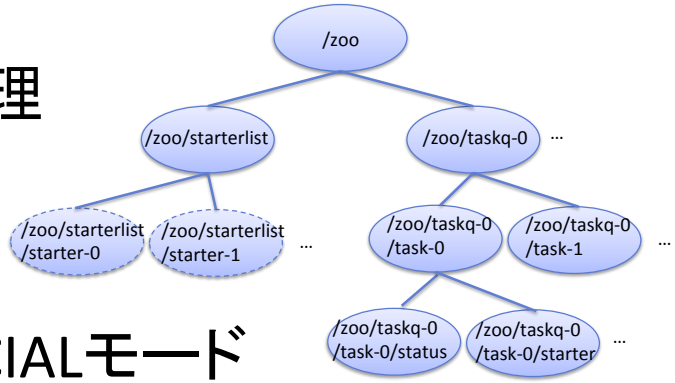


発表概要

- 対象アプリケーションとプログラムイメージ
- 高可用分散協調セルフスケジューリング機構の設計
- Apache ZooKeeperを用いた試験実装
- 関連研究
- まとめと今後の課題

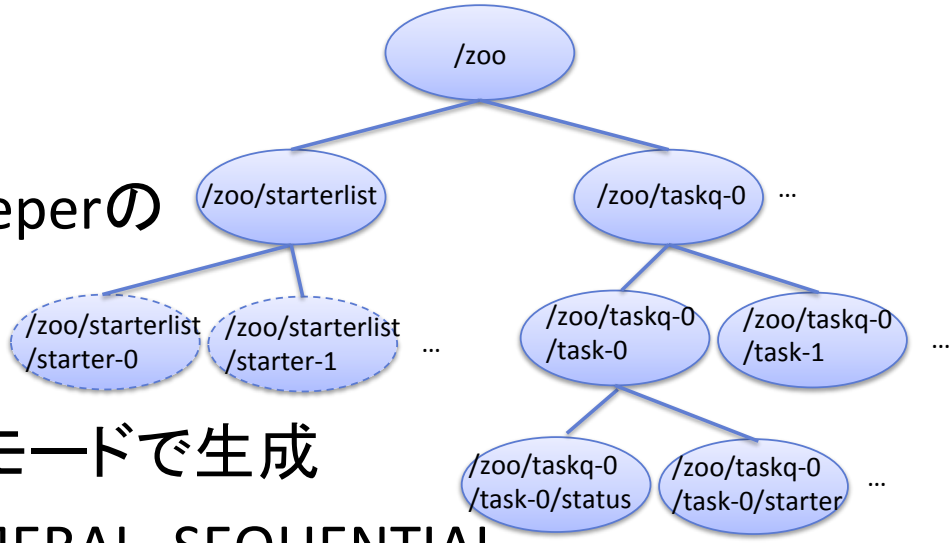
Apache ZooKeeper

- ZooKeeperの高可用性
  - 分散協調処理を支援する機能をライブラリとして提供
  - 複数プロセスのグループ（アンサンブル）で単一障害点回避
  - Paxosをベースとしたアトミックブロードキャストプロトコル Zabで，複数プロセス間でのメッセージ順序を保証
- ZooKeeperのデータモデル
  - znodeと呼ばれるノードのツリーを管理
    - znodeでは1MBまでのデータを扱う
  - アクセス制御リスト（ACL）を設定可能
  - EPHEMERAL / PERSISTENT，SEQUENCIALモード
    - EPHEMERALはznodeの生成者との接続が切れると消失
  - znodeの変化を監視するウォッチを設定可能



スケジューリング機構へのZooKeeperの適用

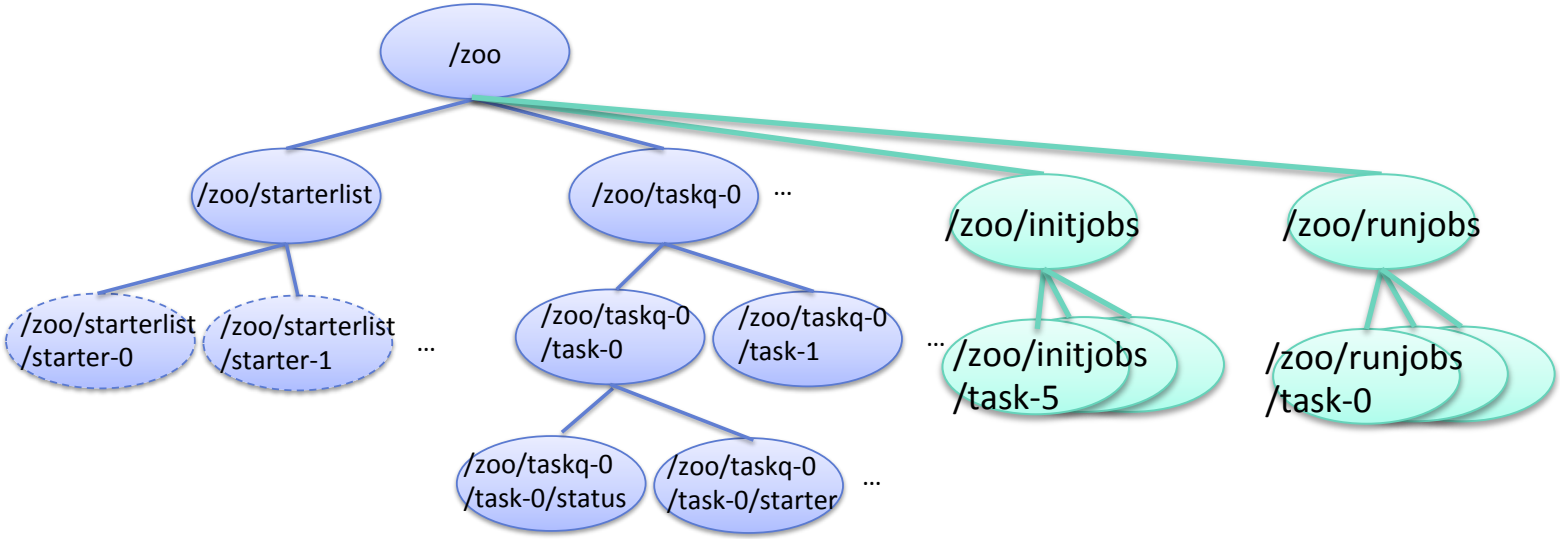
- In-memory FSへの適用
  - In-memory FSとしてZooKeeperのデータモデルを利用
  - タスクキュー情報をPERSISTENT_SEQUENTIALモードで生成
  - Starterリスト情報はEPHEMERAL_SEQUENTIALモードで生成→Starterの障害検知に活用



- WatchDogでのStarterの死活監視への適用
  - 定期的にRUNNING状態かつStarterに故障がある（znodeがない）タスクがあるか確認
  - WatchDogのリスト情報もEPHEMERAL_SEQUENTIALモードで生成し，リーダー選出に活用

実装上の工夫

- タスクキューへのアクセス集中の緩和
  - Starterがタスクの取得に失敗した場合，平均N秒(一様分布)の待ち時間後にリトライ
- インデキシング
  - INITIAL，RUNNINGのジョブのノード情報を別途管理し，Starter，WatchDogでのタスク検索の高速化



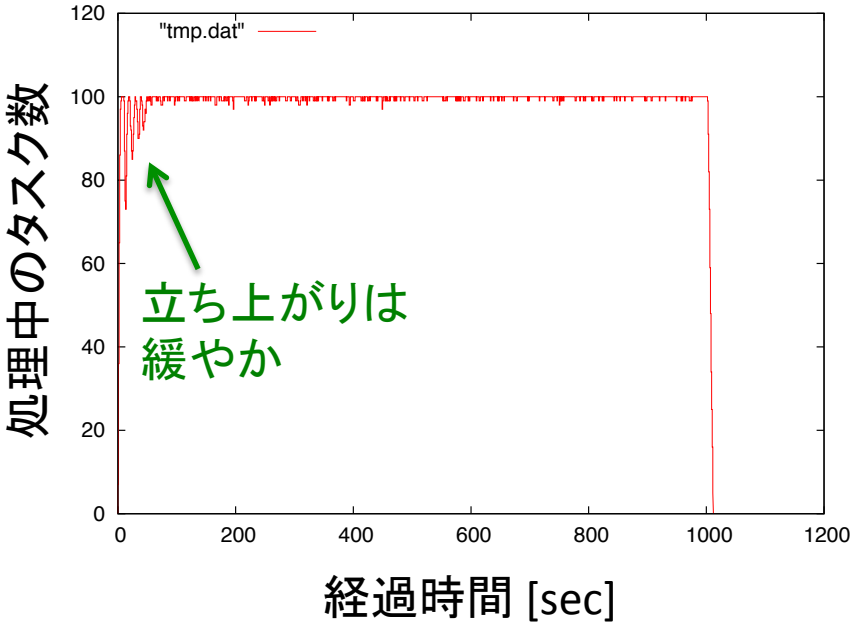
試験実装を用いた実験

- 試験実装の実行確認
  - 正常に実行した場合と障害が発生した場合の挙動を比較
  - 失敗したタスクは再実行
- 実験環境
  - InTrigger Hongo100-110を利用
    - Intel Core2Duo 2.13GHz, Memory 4GB, HDD 500GB, GigE
  - 10x10 Starterで10[sec]x10000タスク（sleep()）を実行

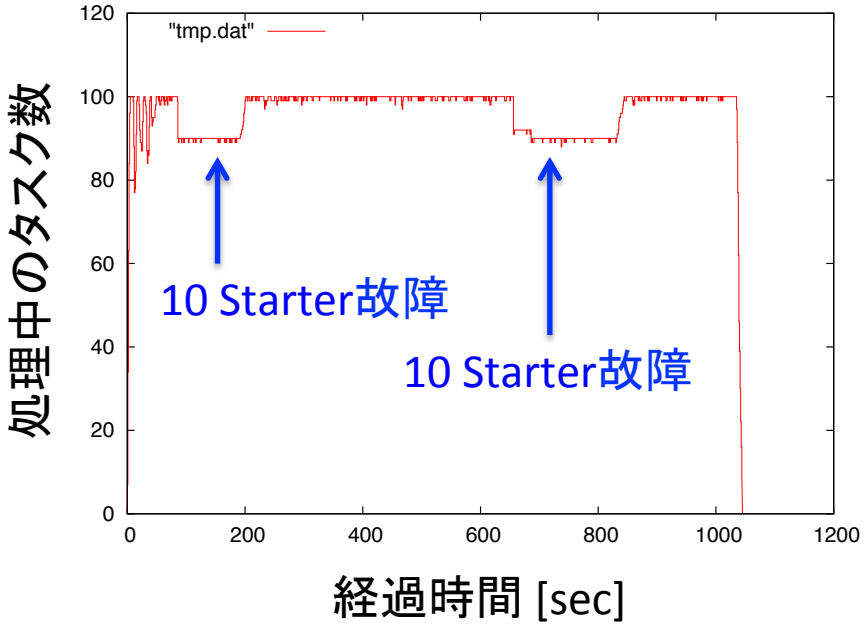
InTrigger Hongo100-110

試験実装を用いた実験の結果

正常実行の場合



障害が発生した場合



故障が発生しても
WatchDogが検知→キューへの再投入-->Starterが再実行
で継続して実行できることを確認

発表概要

- 対象アプリケーションとプログラムイメージ
- 高可用分散協調セルフスケジューリング機構の設計
- Apache ZooKeeperを用いた試験実装
- 関連研究
- まとめと今後の課題

関連研究

- Condor Master Worker [Gouxら，HPDC2000]
  - マスターワーカー型アプリケーションのためのフレームワーク
  - Condorバッチスケジューラでワーカを自動的に計算機に割り当てる
  - タスクの再実行／削除による継続実行，ポストペタスケール計算機環境でのスケーラビリティに課題
- 分散ジョブスケジューリングシステム[梅田ら, HPC2006]
  - 資源情報収集，ジョブと資源のマッチングを広域分散処理
    - ゴシッププロトコルで資源情報共有
  - 疎結合な環境でスループット向上を目的

まとめ

- スケーラブルかつ可用性の高い分散協調セルフスケジューリング機構を提案
  - Submitter：キューへのタスク投入，再実行／削除
  - Starter：キューの先頭タスクを自律的に実行
  - WatchDog：Starterの死活監視
  - In-memory FS：資源管理情報管理，永続化
- Apache ZooKeeperを用いた試験実装
  - 故障が発生しても継続して実行できることを確認

今後の課題

- データストア機構との連携
  - 試験実装では，分散KVSのApache Cassandraの利用を検討
    - データの冗長管理機能，耐故障性を備える
- 資源管理のスケーラビリティ
  - 試験実装ではStarterと計算ノードは1対1
  - 階層的な計算ノードの管理など，より複雑な処理が必要
- 耐障害性機能によるアプリケーション性能への影響の調査

謝辞

本研究の実験では，情報爆発時代に向けたIT基盤技術研究のための研究プラットフォーム"InTrigger"を用いた.