

平成22年度 学士論文

非集中型クラウドストレージの スケーラビリティ評価

東京工業大学 理学部 情報科学科

学籍番号 07-0615-4

奥寺 昇平

指導教員

首藤 一幸 准教授

平成23年2月7日

概要

本稿は、プログラム分散化のためのアスペクト指向言語（AOP）である Addistant を拡張した、Addistant 2 を提案する。Addistant は、利用者が分散に関する記述を独自のアスペクト指向言語で指定することにより、単一の Java 仮想機械（JVM）上で実行することを目的として開発した既存プログラムを、利用者が望む形で機能分散化を導入する。この分散化はロード時に、バイトコードレベルで行われる。Addistant 2 は、分散アスペクトの記述力が十分ではなかった Addistant を、分散アスペクトの Join point の種類を増加することにより、大幅に分散アスペクトの記述力を強化した。また、本稿は Addistant 2 の典型的な利用方法の例も示す。

謝辞

本稿は以下の方々なくして、存在しえなかったでしょう。Addistant の開発、Addistant 2 の提案および本稿の編集になにかと心を砕いていただいた千葉滋講師、東京大学の光来健一氏、筑波大学の立堀道昭氏、横田大輔氏そして研究室のみなさん。心より感謝しています。

(具体的に何をしてもらったか書く)

目次

第 1 章	はじめに	6
第 2 章	関連研究	8
2.1	Javassist	8
2.2	Addistant	8
第 3 章	測定手法	9
3.1	Cassandra の概要	9
3.2	Cassandra の軽量化	9
3.2.1	プログラムの改変	9
3.2.2	設定ファイルのパラメータ調整	9
3.3	実験シナリオ	9
3.3.1	実験環境	9
3.3.2	計測方法について説明	9
3.4	通信量の測定について	9
第 4 章	実装	10
付 録 A	プログラム例	11

図 目 次

4.1 Javassist の処理の流れ	10
--------------------------------	----

表 目 次

第1章 はじめに

今日、分散ソフトウェア、つまり複数の計算機上で動作するソフトウェアの必要性が高まる一方、その開発にかかるコストが問題になっている。これは、分散プログラムを作成する場合にネットワークなどの分散環境特有の問題に対処しなければならないためである。それらの処理の記述を含んだ分散プログラムは煩雑になり、非分散プログラムの作成に比べて、分散プログラムの作成や維持にかかる人的コストは飛躍的に大きくなりがちである。

分散プログラミングが煩雑であることの要因として、プログラムの分散に無関係な記述の中に分散に関わる処理が拡散して入り交じっている (crosscutting concerns) ことがあげられる。このようなプログラムは可読性が低く、変更も大変である。分散に無関係な記述が分かりにくくなる上、分散に関わる処理を変更するためにはプログラムのあちこちを修正しなければならない。

我々は分散プログラミングをアスペクト指向言語 (AOP) により支援する Addistant を開発してきた。ここで、アスペクト指向とは、オブジェクト指向との相補性を意識した概念である。オブジェクト指向とは、機能性という点に着目して全体をオブジェクトと呼ばれるモジュールに分割していく概念である。しかし、個々のモジュール内には、同様の処理が股がる可能性がある。この個々のモジュールに股がった同様な処理を、オブジェクトとは別の側面から考慮し、それをモジュール化する概念を**アスペクト指向**といい、そのモジュールを**アスペクト**という。

Addistant は次の特徴をもつ。

- Addistant の利用者に、各クラスごとそのオブジェクトを分散環境中のどこに配置するかを指定させる。Addistant では、分散に関わる処理がプログラム全体に拡散して入り交じることを避けるため、利用者は、分散に無関係な記述である非分散プログラムとは別に、分散に関わる記述をまとめて記述する。このまとめて別に書かれた分散に関わる記述を**分散アスペクト**と呼ぶ。
- 対象プログラムのバイトコードを変換して、分散アスペクトによって指定されたクラスが、遠隔ホストで動作している Java 仮想機械 (JVM) 上で実行されるようにする。Addistant は変換のために、対

象プログラムのソースコードを必要としない。変換されたバイトコードは正規の Java バイトコードであり、実行のために特別な JVM を必要としない。バイトコード変換には Javassist を用いた。

- 変換されたバイトコードを Addistant の実行系により遠隔 JVM に自動的に配布される。

Addistant では、遠隔オブジェクト参照を、従来の分散プログラミング・ツールで使われてきたアイディアを組み合わせで実現した。この実装は、**プロキシ・マスタ方式**に基づいたもので、Addistant がバイトコード変換により自動的に行う。

しかしながら、Addistant では、開発が進むにつれ、その利点とともに問題点も明らかになってきた。その問題点とは、分散アスペクトの記述力が十分ではなかったため、利用者が望む機能分散をうまく実現できない場合があることである。

そこで我々は、Addistant の分散アスペクトの記述力を大幅に高めた Addistant 2 を開発中である。本稿では Addistant 2 の典型的な利用方法の例をあげ、分散アスペクトの新しい記述方法を説明する。

本稿の残りは、次のような構成からなっている。(以下では、Addistant を Addistant 1 と呼ぶ。) 第 2 章は 分散プログラミング用アスペクト言語の必要性和 Addistant 1 の性能を述べる。第 3 章では、Addistant 2 の分散アスペクト記述を、第 4 章では、Addistant 2 のこれまでの実装、第 5 章では、Addistant 2 を利用したソフトウェアの機能分散例、第 6 章では、関連研究を、そして第 7 章でまとめを述べる。

第2章 関連研究

2.1 Javassist

Javassist [?, ?] は...

Javassist は [?] で配布されている。

2.2 Addistant

第3章 測定手法

gosi

3.1 Cassandraの概要

・メンバーシップ周り？・seedの説明・データ保管部分の解説・ある特定のポートで通信を行なっている。・cassandraは、IPアドレスのみでノードを判定している。

3.2 Cassandraの軽量化

実験に当たって、物理リソースからの都合上1台あたり複数のCassandraノードを起動する必要があった。デフォルトの設定で、オリジナルのCassandra一台起動するには、データをまったく入れない状態で、スレッド数が130程度、メモリー150M程度消費する。多数のノードを立ち上げるためにデータ保持部分のプログラムの改変と、設定ファイルのパラメータ調整を行った。

3.2.1 プログラムの改変

Cassandraでは、何もデータが入れない状態であっても、Systemなんちゃらを各ノードで保持すること、時間が経つと徐々にOnmemory上にmemtable?とかが増えてしまことで、メモリー使用量域がかさむ。私の実験に関しては、実データがどのようなものかは関係がないので、このようなデータを小さく保持する必要があった。そこで、実際のデータを保存するのではなく、データサイズだけを保管するように変更した。その結果メモリー使用量域がへった。(具体的には、絶対的に減っていくつになった。2.時間がたっても増えにくいことをアピール)

3.2.2 設定ファイルのパラメータ調整

また、設定パラメータを調節した。Cassandra は、write、read 時のスレッド数を制限することでスレッド数を 130 から 100 に落とした。

3.3 実験シナリオ

実験では一台あたり複数の Cassandra ノードを起動した 10 台のマシンを利用し、通信量の測定を行った。各 Cassandra の起動、通信量の測定は、シェルスクリプトでプログラムを書き制御した。

3.3.1 実験環境

以下に実験環境を示す。

- Cassandra 0.6.6
- OS: Linux 2.6.35.10-74.fc14.x86_64 Java 仮想マシン : *JavaSE6Update21*
- CPU: 2.40 GHz Xeon E5620 × 2
- メモリー: 32GB RAM
- ネットワーク: 1000BASE-T

3.3.2 計測方法について

- ・ ipaliasling をかけてネットワークを構成したこと

Cassandra では、IP アドレスだけでノードを管理している。つまり、今回のように、一台あたり複数台の Cassandra を単純に立ち上げると、不都合が生じる。そこで、IPAliasling を使って、仮想アドレスを作成し、Cassandra ノードごとに割り振った。これにより、同じマシン上に立ち上がった Cassandra マシンが判別でき、不整合が起きなくなった。通信量の測定の際には、ノイズを防がないといけない。ノイズとは、Cassandra ノード以外から要求されるリクエストのことである。具体的には、ARP とか、PING とかある。これらを誤って計測することを避けるために IPAliasling を行うと同時に、CassandraNode に割り振られた IP アドレスだけが含まれる private network を構築し、ノイズを減らした。(なくした?) (IPAlias の図でも載せておく ???)

- ・ tcpdump をしよう。

通信量の測定は、tcpdump を使用した。上述したように、Cassandra Node

同士のやりとりは、プライベートネットワーク上で行われるため、このネットワークをまたぐ TCP パケットのサイズを記録した。(具体的には、とか書いとく?) tcpdump 10.20.....

このコマンドを各マシンで実行することで、外部のマシン上で実行されている CassandraNode からの通信を計測することができる。

3.4 通信量の測定について

しかしながら、この計測方法では同じマシン上での CassandraNode 同士の通信を計測することはできない。だが、計測した総通信量から Cassandra Node で発生する総通信量を計測できることを示す。

ここで、一つ仮定を立てる。

・任意の Cassandra Node 同士の通信量は平均すると同じ!(言葉をおきかえる!)

この仮説をもとにすると、n 台の各マシンで m 台の cassandra node を起動したとする。(つまり、合計 $n \times m$ 台の Cassandra ノードを立ち上げたとする) さらに、上のように、各マシンで tcpdump を使用して計測した得られたトラフィックの合計を T とおき、Cassandra node で発生するそう通信量 TT とく。 $(n-1) \times m$ 台の cassandra node で発生している通信量が T であるので、仮説を使うと、 $n \times m$ 台の場合は、

$$TT = T * (n * m) / ((n-1) * m) = T * n / (n-1)$$

となる。具体的に、 $n = 10$ 台の時について、図を使いながら解説する。(図を使おう)

第4章 実装

図 4.1 は...

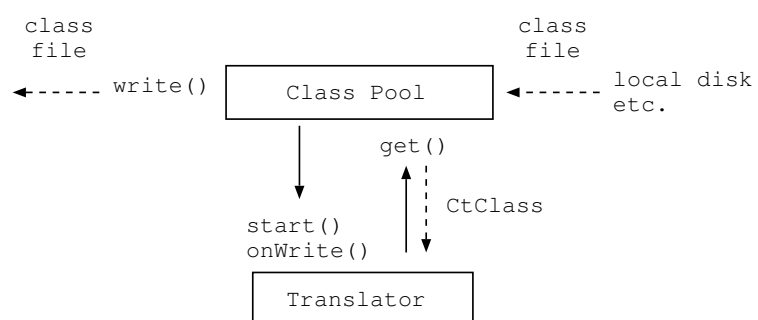


図 4.1: Javassist の処理の流れ

付 録 A プログラム例