Hadoopのノード脱退時および削除時のレプリカ生成に関する一考察

日開　朝美†　　竹房あつ子††　　中田　秀基††　　小口　正人†

† お茶の水女子大学　〒 112-8610 東京都文京区大塚 2-1-1
†† 産業技術総合研究所　〒 305-8568　茨城県つくば市梅園 1-1-1
E-mail: †asami@ogl.is.ocha.ac.jp, ††{atsuko.takefusa,hide-nakada}@aist.go.jp, †††oguchi@computer.org

あらまし　大規模データに対応した処理システムとして，汎用的なハードウェアを用いて高度な集約処理を行う分散ファイルシステムに注目が集まっている．本研究では，Apache Hadoop の基盤技術である Hadoop Distributed File System(HDFS) に着目した．Hadoop はスケールアウトするシステムであり，システム規模に応じた運用や故障の発生などにより，ノードを脱退もしくは削除することが想定される．HDFS は通常複数の DataNode 上に複数のレプリカを保持し，ノード脱退時および削除時にはレプリカを自動的に生成する．この過程が長くなると一部の DataNode に負荷がかかり性能が低下してしまうため，その高速化が重要である．本稿では，ノード脱退時および削除時の不足分を補うレプリカ生成処理に関するスループットを評価する．また，レプリカ生成を高速化するために，レプリカ生成先を制御する手法を提案する．制御手法により約 10MB/sec 程スループットが向上したこと，およびレプリカ生成処理が若干効率化したことを示す．

キーワード　分散ファイルシステム，HDFS，レプリカ生成，ノード削除，ノード脱退

A Study on Replica Generation at Node Decommission and Deletion for Hadoop

Asami HIGAI†, Atsuko TAKEFUSA††, Hidemoto NAKADA††, and Masato OGUCHI†

† Ochanomizu University　　2-1-1 Otsuka, Bunkyouku Tokyo 112-8610 JAPAN
†† National Institute of Advanced Industrial Science and Technology (AIST)　1-1-1 Umezono, Tsukuba, Ibaraki,305-8568, Japan
E-mail: †asami@ogl.is.ocha.ac.jp, ††{atsuko.takefusa,hide-nakada}@aist.go.jp, †††oguchi@computer.org

1. はじめに

近年，通信技術や情報処理技術の発展と普及に伴い，データ量が爆発的に増加している．そこで汎用のハードウェアを用いて高度な集約処理を行う分散ファイルシステムに注目が集まっている．分散ファイルシステムは，高いスケーラビリティを持つため，システムの規模に応じた運用が見込まれる．また，従来では想定されていなかったような極めて大規模なシステム構成を取る機会が増えた．しかしながら，全てのマシンを正常に動作させることは難しく，常に一定の割合で故障や動作不安定なマシンが存在してしまうという問題が新たに生じる．

Hadoop Distributed File System(以下 HDFS) [1] は，Apache Hadoop(以下 Hadoop) [2] の基盤技術であり，Google の分散ファイルシステム GFS [3] に基づいて設計された．Hadoop はスケールアウトするシステムであり，システム規模に応じた運用や故障の発生などにより，ノードを脱退もしくは削除する事が想定される．HDFS では通常複数の DataNode 上に複数のレプリカを保持し，ノード脱退時および削除時にはレプリカを自動的に生成する．この過程が十分に早くないと一部の DataNode に負荷がかかり性能が低下してしまうためその高速化が重要である．

本稿では，HDFS においてノード脱退時および削除時に不足分を補うレプリカ生成処理の特性を把握するためにスループットを測定した．またレプリカ生成の高速化のために，レプリカ生成先を制御する手法を提案して，評価実験を行った．制御手法により約 10MB/sec 程スループットが向上したこと，およびレプリカ生成処理が若干効率化したことを示す．

2. 分散ファイルシステム

2.1 分散ファイルシステム

本研究では，汎用のハードウェアを用いて高度な集約処理を行う分散ファイルシステムに注目した．分散ファイルシステム

は，ハードウェアの制約が無く，高いスケーラビリティを持つため，システム規模に応じた運用が可能である．また多数の計算機が並列分散処理を行うため，大規模データに対して効率良く処理を行うことができる．

2.2 Hadoop Distributed File System

Hadoop は Apache Software Foundation で開発されている分散コンピューティング関連のプロジェクト群で，様々なサブプロジェクトの集合体である (図 1)．HDFS はそのサブプロジェクトの一つで，Google の分散ファイルシステム GFS に倣って設計されたオープンソースの分散ファイルシステムである．HDFS は，ファイルのメタデータやクラスタ内のノード管理を行う一台の NameNode と，実際にデータを格納している複数台の DataNode からなる．ファイルを最小単位であるブロックに分割して DataNode 間で分散して保存することで，耐故障性と対多クライアントでの高いパフォーマンスを維持している．

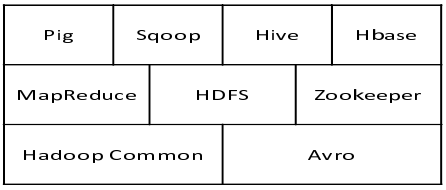

図 1　Hadoop のサブプロジェクト（2012 年 5 月現在）

3. ノード脱退時および削除時のレプリカ生成の流れ

3.1 ノード脱退とノード削除

HDFS はクラスタからノードを切り離す際，該当ノードが保持しているデータのレプリカを残りのノードに移行することで，指定したレプリカ数を維持する．本研究ではクラスタからノードを切り離す方法として，ノード脱退とノード削除の二種類が存在する．

ノード脱退とは，事前に脱退ノードが保持しているデータのレプリカを他のノードに複製してからそのノードをクラスタから切り離す方法である．クラスタ規模の縮小やエラー率が非常に高いノードが存在する時など，意図的にクラスタからノードを切り離す場合を想定している．

ノード削除とは，クラスタからノードが離脱後，残ったノード間で削除ノードが保持していたデータのレプリカを複製し補完する方法である．ノードの故障や動作不良あるいはネットワークの切断等により，想定外にノードが突然クラスタから離脱してしまう場合を表す．

3.2 レプリカ生成の流れ

ノード脱退もしくは削除処理が実行されると，その DataNode で管理されていたレプリカの再配置のためにレプリカ生成処理が行われる．レプリカ生成はブロック単位に処理が進み，まず NameNode が定期的にレプリカ生成の指示をそれぞれの DataNode に転送する．DataNode は受けとった指示をもとに，レプリカ生成先の DataNode へデータの転送を始める．生成先でデータの複製が完了すると，生成元の DataNode に生成完了の ACK を転送する．この時 NameNode が定期的に DataNode に指示する転送間隔は変数 `replicationRecheckInterval` で定義され，転送レプリカブロック数は，クラスタに接続されている DataNode 数*変数 `REPLICATION_WORK_MULTIPLIER_PER_ITERATION` で定義される．DataNode が生成完了の ACK を受け取らない状態のまま，レプリカ生成先の DataNode へ転送できるブロック数は変数 `dfs.max-repl-stream` で定義される．`replicationRecheckInterval` と `REPLICATION_WORK_MULTIPLIER_PER_ITERATION` は，FSNamesystem.java の ReplicationMonitor クラスで定義されている変数であり，`dfs.max-repl-stream` はプロパティで変更可能な変数である．各変数のデフォルト値は表 1 のようになっている．

表 1　各変数のデフォルト値

変数	デフォルト値
replicationRecheckInterval	3
REPLICATION_WORK_MULTIPLIER_PER_ITERATION	2
dfs.max-repl-stream	2

4. 基本性能評価

4.1 実験環境

本実験ではレプリカ生成に関する評価を行うために，ノードを切り離す方法の違いと，ブロックサイズおよび同時に処理するデータ量の変化がレプリカ生成処理のスループットに与える影響を調査する．ブロックのレプリカ数を 3 とし，1 台の DataNode をクラスタから切り離す際のスループットを示す．ここでスループットは以下のように定義した．

$$\text{スループット [MB/sec]} = \frac{\text{脱退及び削除ノードが保持しているデータ量 [MB]}}{\text{データの複製が開始してから完了するまでの時間 [sec]}}$$

ローカルクラスタ上で Hadoop-1.0.3 をインストールしたマシン 7 台を用いた．そのうち 1 台を NameNode とし，残りの 6 台を DataNode とした．マシンのスペックは全て同一で表 2 に示す通りである．

表 2　マシンスペック

OS	Linux 2.6.32-5-amd64 Debian GNU/Linux 6.0.4
CPU	Quad-Core Intel(R) Xeon(R) CPU @ 1.60GHz
Main Memory	2GB
HDD	73GB SAS × 2(RAID0)
RAID Controller	SAS5/iR

4.2 ノード脱退およびノード削除の基本性能

ブロックサイズをデフォルトで設定されている 64MB とし，ノード脱退とノード削除の 2 つの方法でクラスタから DataNode を切り離した際の，スループットを測定した結果を図 2 に示す．図 2 より，ノード脱退時の方がノード削除時に比べてスループットが高いことが分かる．これはノード削除時には，削除ノードを除く残りのノードでレプリカ生成処理が行われるのに対して，ノード脱退時には，脱退ノード自身がレプリカ生成元としてレプリカ生成処理に参加することができるためである．

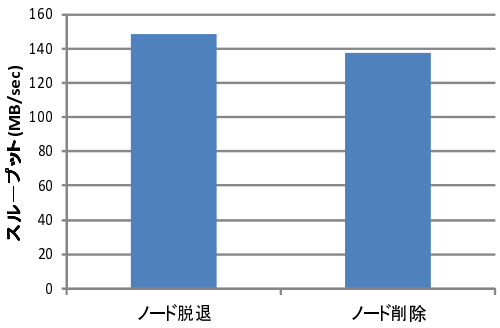

図 2　ノード脱退時とノード削除時のスループット

4.3　同時転送ブロック数を変化させた実験

ブロックサイズ 16，32，64，128，256MB に対して，レプリカ生成時の NameNode と DataNode の同時転送ブロック数に関連する，`REPLICATION_WORK_MULTIPLIER_PER_ITERATION` と `dfs.max-repl-stream` の 2 つの変数を共に同じ値 α とし，α を 1〜8 に変化させてノード削除を行った際のスループットを図 3 に示す．図 3 より，α が小さいとき，すなわち同時転送ブロック数が少ない場合，ブロックサイズが小さい時にはスループットが低くなる．これはブロックサイズが小さいと DataNode はレプリカ生成処理が早く完了して，NameNode から定期的に送られてくるレプリカ生成の指示を待っている状態が生じるからと考えられる．そのためこの状態にある間は，α に比例してほぼ線形にスループットが増加している．またこの指示待ち状態が解消された後は，α の増加，すなわち同時転送ブロック数の増加に伴い，スループットが若干低下している．多数のスレッドが生成されたために，ディスクへの書き込みや通信の衝突によるオーバヘッドが発生しているためと考えられる．

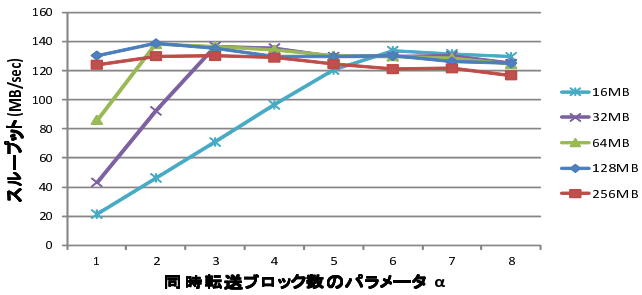

図 3　各ブロックサイズにおける
同時処理するブロック数に応じたスループット

5.　レプリカ生成先を考慮した制御手法の提案

HDFS では，ノードが同一ラックに配置されている場合，レプリカの生成先は該当のレプリカを保持していないノードからランダムに選出される．一見レプリカの生成先が分散し効率が良さそうだが，分散させた結果，一対多の通信が多発してしまうことや，生成先の選出はランダムでありながらも，偶発的にあるノードに集中してしまうことにより，idle 状態のノードが存在してしまうことが考えられる．これらは通信効率を悪化させる要因であり，これらを改善することでレプリカ生成が効率的に行われることが期待できる．そこでレプリカ生成の高速化に向けて，まずはレプリカ生成先を制御する手法を提案し，制御前後でスループットを比較して有効性を検証する．

5.1　提 案 手 法

Dolly [4] や catwalk-ROMIO [5] で用いられてるように，ノードをリング状に配置し，そのリング構造に従って一方向にデータ通信を行うと，ネットワーク内で通信のボトルネックとなるデータの集中が避けられるため効率が良いことが知られている．そこで本提案手法では，ノードがリング状に配置されていると仮定し，レプリカ生成先を以下のように定義する (図 4)．

(1) レプリカ生成先をリング上の次のノードに指定する．
(2) もしそのノードがすでにレプリカを持っている場合は，さらに次のノードをレプリカ生成先にする．

このようにレプリカ生成先を制御することで，データの流れを一方向に制御し，多数のノードと通信する事を回避して，通信効率を向上させることを目指す．

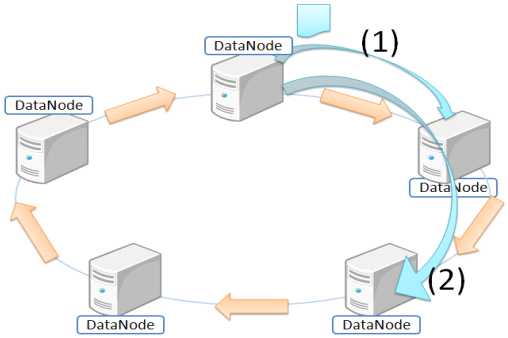

図 4　提案手法のレプリカ生成先

5.2　提案手法の評価

4.3 節と同様の実験を提案手法に関して行い，ブロックサイズ毎の制御前後のスループットの測定結果を図 5 に示す．図 5 より，DataNode が生成完了の ACK を受け取らない状態のまま，レプリカ生成先の DataNode に転送できるブロック数以上のレプリカ生成の指示を NameNode から受け取っている時には，全ての場合で提案手法のスループットが 10MB/sec 前後向上している．16，32，64MB において α が小さい時には，制御前後のスループットに変化がないが，これは 4.3 節で述べたように DataNode は NameNode からのレプリカ生成の指示を待っている状態であると考えられることから，スループットが向上する余地が無いため妥当な結果である．また提案手法は，ブロックサイズが大きいほど有効であることが確認できる．ブロックサイズが大きい方が 1 つのブロックのレプリカ生成にかかる時間が長く，複数ノード間での通信効率を向上させる提案手法の効果が大きいためである．

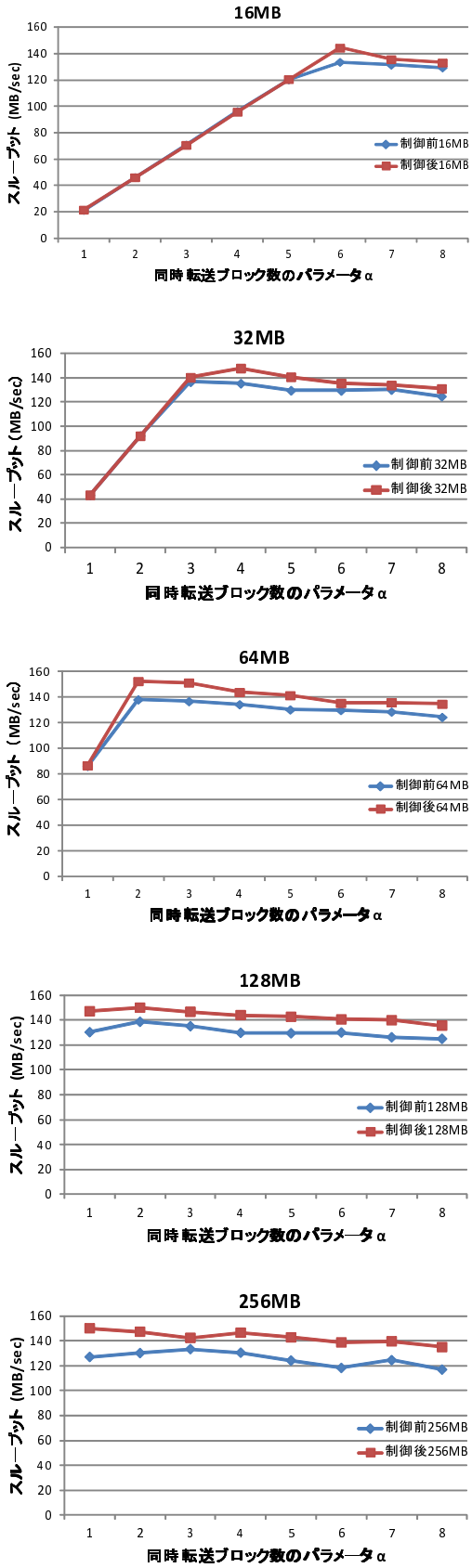

図 5　ブロックサイズ毎の制御前後のスループット

通信効率の向上を目指す提案手法による通信効率の向上および，制御前後のスループットの変化を示すために，制御前後の各 DataNode の 1 秒毎の送受信のスループットをヒストグラムにした．ここではデータ数が多いため，デフォルトの設定である 64MB，α=2 のときと，ブロックサイズおよび同時転送ブロック数が多い場合の 256MB, α＝8 の時のヒストグラムをそれぞれ図 6 と 7 に示す．実験毎にレプリカ生成処理の実行時間が若干異なり，また一回の実験においても各 DataNode の処理の終了時刻も異なることから，データはレプリカ生成処理の開始 1 秒から 100 秒までを扱っている．一回の実験のデータ数は，100 秒× DataNode5 台の計 500 個であり，図 6 と 7 は送信 (上) と受信 (下) の 3 回分の実験結果をまとめたもので，データ数は，それぞれ 500 個×実験 3 回の計 1500 個である．横軸がスループット (MB/sec) のデータ区間で，0，(0，10]，(10，20]，…と 10 刻みとし，縦軸が各データ区間のデータ数である．

ブロックサイズが 64MB，α=2 と扱うデータが比較的小さいとき，図 6 より，送信の結果では，制御前に比べて制御後のデータ区間 0 のデータ数が増加し，ヒストグラムのピーク値が右にシフトしている．データ区間 0 の増加は，通信をしていない時間が増加したということである．提案手法は生成先を隣のノードに制御しているため，処理が均一に分散しやすく，また多数のノードとの通信を回避できるため，通信効率が上がり，4.3 節で述べたように各 DataNode で NameNode からの指示待ち状態が増えたためであると考えられる．またヒストグラムのピーク値が右にシフトしているのは，通信効率の向上に起因するものと考えられる．

受信の結果では，制御前に比べて制御後のデータ区間 0 のデータ数が減少していること，およびデータ区間 60，70 が増加していることがわかる．データ区間 0 の減少は，通信をしていない時間が減少したということであり，送信とは逆の結果であるが，これは提案手法により処理が均一に分散しやすくなったためデータを受信しない機会が減少したためだと考えられる．またデータ区間 60，70 が増加したのは，通信効率が向上したためであると考えられる．

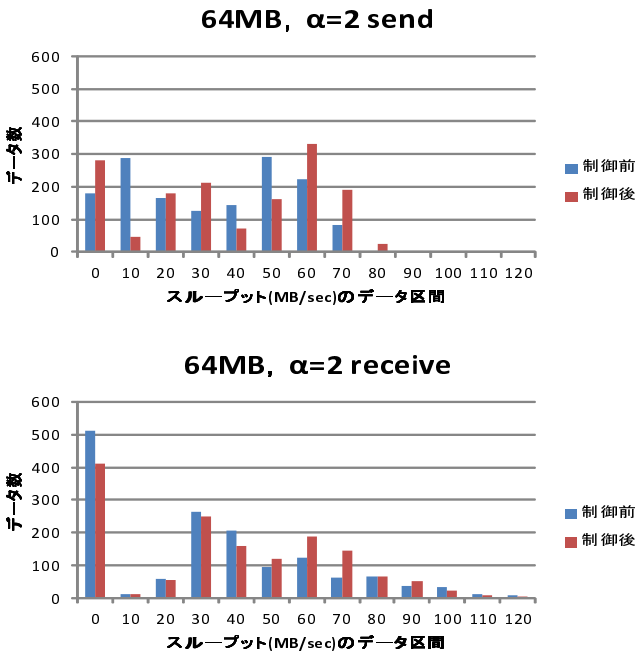

図 6　64MB，α=2 の時の送受信のヒストグラム

ブロックサイズが 256MB，α＝8 と扱うデータが大きいとき，図 7 より，データ区間 0 のデータ数が送受信ともに 0 かそれに近い値であることが分かる．これはブロックサイズが大きく，同時転送ブロック数も多いため，常に通信が継続して行われているためである．また送受信ともに，ヒストグラムのピーク値が右にシフトしている．これもまた通信効率が向上したためであると考えられる．

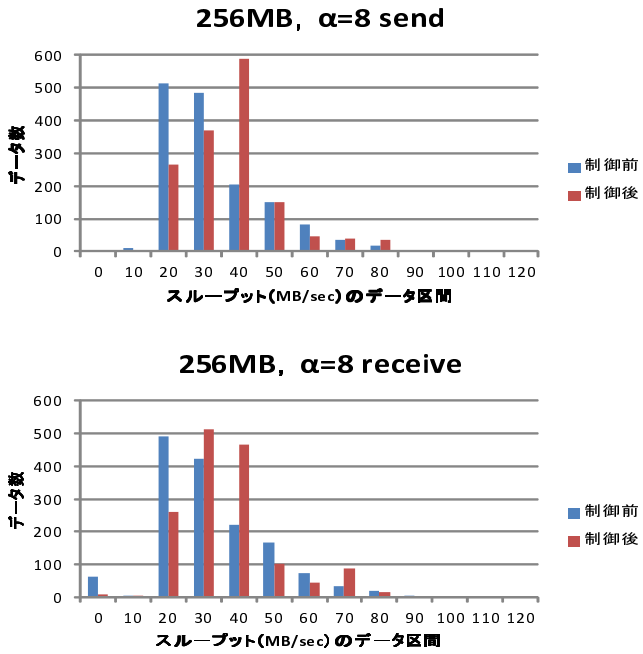

図 7　256MB，α＝8 の時の送受信のヒストグラム

6.　まとめと今後の課題

分散ファイルシステムの一つである HDFS 上で，ノード脱退時とノード削除時のレプリカ生成に関する基本性能の評価を行った．ノード脱退時の方が削除時に比べてスループットが高いことと，レプリカ生成処理のブロックサイズが小さく，同時転送ブロック数も少ないとスループットが低いこと，および同時転送ブロック数を増加させると，処理のオーバヘッドによりスループットが低下することを確認した．またレプリカ生成の高速化のためにレプリカ生成先を制御する手法を提案し，実装した．評価実験から制御手法によりスループットが 10MB/sec 程向上した．また，制御手法により若干であるがデータが均一に分散されて，効率の良い処理が実現できた．

今後の課題としては，本稿のレプリカ生成先の制御に加えて，生成元の制御も行い，更なるレプリカ生成の高速化を目指したい．また HDFS では複数のレプリカは，信頼性と読み書きに必要な帯域のトレードオフを考慮し，異なるラックに分散して配置される．本実験環境は全てのノードが同一のラック上にあるためラックを考慮しないレプリカ配置を行っていたので，より現実的環境を想定し，ラックを意識した制御にも取り組んでいきたい．

文　　献

[1] Dhruba Borthakur, ”HDFS Architecture,” 2008 The Apache Software Foundation.

[2] Tom White(著), 玉川竜司, 兼田聖士 (訳):Hadoop, オライリー・ジャパン,2010

[3] Ghemawat, Sanjay; Gobioff, Howard; Leung, Shun-Tak (October 2003), ”The Google File System,” 19th Symposium on Operating Systems Principles (conference), Lake George, NY: The Association for Computing Machinery, CiteSeerX: 10.1.1.125.789, retrieved 2012-07-12.

[4] Felix Rauch, Christian Kurmann, Tomas M.Stricker, ”Partition Repositories for Partition Cloning — OS Independent Software Maintenance in Large Clusters of PCs,” Proceedings of the IEEE International Conference on Cluster Computing 2000, Chemnitz, Germany, Nov 28 - Dec 2, 2000.

[5] 堀敦史，鴨志田良和，松葉浩也，安井隆，住元真司，石川裕：ファイルステージングシステム Catwalk の MPI-IO 実装，情報処理学会研究報告，2009 年 8 月．