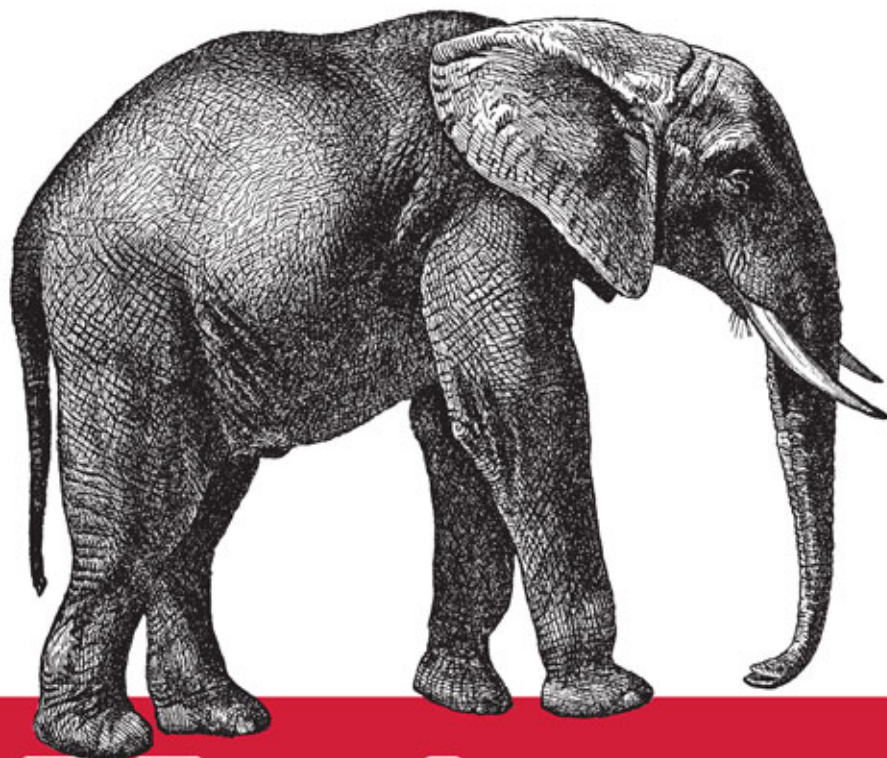


O'REILLY®



# Hadoop

Kompletny przewodnik

ANALIZA I PRZECHOWYWANIE DANYCH

Helion 

Tom White

Tytuł oryginału: Hadoop: The Definitive Guide, Fourth Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-283-1457-3

© 2016 Helion SA

Authorized Polish translation of the English edition of Hadoop: The Definitive Guide, 4th Edition ISBN 9781491901632 © 2015 Tom White.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:  
<ftp://ftp.helion.pl/przyklady/hadoop.zip>

Drogi Czytelniku!  
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres  
<http://helion.pl/user/opinie/hadoop>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

---

# Spis treści

<b>Przedmowa</b> .....	<b>17</b>
<b>Wprowadzenie</b> .....	<b>19</b>
Kwestie porządkowe .....	20
Co nowego znajdziesz w wydaniu czwartym? .....	20
Konwencje stosowane w tej książce .....	21
Korzystanie z przykładowego kodu .....	21
Podziękowania .....	22
<b>Część I. Podstawy platformy Hadoop</b> .....	<b>25</b>
<b>Rozdział 1. Poznaj platformę Hadoop</b> .....	<b>27</b>
Dane! .....	27
Przechowywanie i analizowanie danych .....	29
Przetwarzanie w zapytaniach wszystkich danych .....	30
Poza przetwarzanie wsadowe .....	30
Porównanie z innymi systemami .....	31
Systemy RDBMS .....	32
Przetwarzanie sieciowe .....	33
Przetwarzanie z udziałem ochotników .....	34
Krótka historia platformy Apache Hadoop .....	35
Zawartość książki .....	38
<b>Rozdział 2. Model MapReduce</b> .....	<b>41</b>
Zbiór danych meteorologicznych .....	41
Format danych .....	41
Analizowanie danych za pomocą narzędzi uniksowych .....	42
Analizowanie danych za pomocą Hadoopa .....	44
Mapowanie i redukcja .....	44
Model MapReduce w Javie .....	45
Skalowanie .....	51
Przepływ danych .....	51
Funkcje łączące .....	55
Wykonywanie rozproszonego zadania w modelu MapReduce .....	56

Narzędzie Streaming Hadoop .....	57
Ruby .....	57
Python .....	59
<b>Rozdział 3. System HDFS .....</b>	<b>61</b>
Projekt systemu HDFS .....	61
Pojęcia związane z systemem HDFS .....	63
Bloki .....	63
Węzły nazw i węzły danych .....	64
Zapisywanie bloków w pamięci podręcznej .....	65
Federacje w systemie HDFS .....	65
Wysoka dostępność w systemie HDFS .....	66
Interfejs uruchamiany z wiersza poleceń .....	68
Podstawowe operacje w systemie plików .....	69
Systemy plików w Hadoopie .....	70
Interfejsy .....	71
Interfejs w Javie .....	74
Odczyt danych na podstawie adresu URL systemu Hadoop .....	74
Odczyt danych za pomocą interfejsu API FileSystem .....	75
Zapis danych .....	78
Katalogi .....	80
Zapytania w systemie plików .....	80
Usuwanie danych .....	84
Przepływ danych .....	85
Anatomia odczytu pliku .....	85
Anatomia procesu zapisu danych do pliku .....	87
Model zapewniania spójności .....	90
Równoległe kopiowanie za pomocą programu distcp .....	91
Zachowywanie równowagi w klastrach z systemem HDFS .....	92
<b>Rozdział 4. System YARN .....</b>	<b>95</b>
Struktura działania aplikacji w systemie YARN .....	96
Żądania zasobów .....	97
Czas życia aplikacji .....	97
Budowanie aplikacji systemu YARN .....	98
System YARN a implementacja MapReduce 1 .....	99
Szeregowanie w systemie YARN .....	101
Dostępne programy szeregujące .....	101
Konfigurowanie programu szeregującego Capacity .....	103
Konfigurowanie programu szeregującego Fair .....	105
Szeregowanie z opóźnieniem .....	109
Podejście Dominant Resource Fairness .....	109
Dalsza lektura .....	110
<b>Rozdział 5. Operacje wejścia-wyjścia w platformie Hadoop .....</b>	<b>111</b>
Integralność danych .....	111
Integralność danych w systemie HDFS .....	112
Klasa LocalFileSystem .....	112
Klasa ChecksumFileSystem .....	113

Kompresja .....	113
Kodeki .....	114
Kompresja i podział danych wejściowych .....	118
Wykorzystywanie kompresji w modelu MapReduce .....	120
Serializacja .....	122
Interfejs Writable .....	123
Klasy z rodziny Writable .....	125
Tworzenie niestandardowych implementacji interfejsu Writable .....	132
Platformy do obsługi serializacji .....	137
Plikowe struktury danych .....	138
Klasa SequenceFile .....	138
Klasa MapFile .....	145
Inne formaty plików i formaty kolumnowe .....	146

## **Część II. Model MapReduce ..... 149**

### **Rozdział 6. Budowanie aplikacji w modelu MapReduce ..... 151**

API do obsługi konfiguracji .....	151
Łączenie zasobów .....	152
Podstawianie wartości zmiennych .....	153
Przygotowywanie środowiska programowania .....	154
Zarządzanie konfiguracją .....	155
GenericOptionsParser, Tool i ToolRunner .....	158
Pisanie testów jednostkowych za pomocą biblioteki MRUnit .....	161
Mapper .....	161
Reduktor .....	164
Uruchamianie kodu lokalnie na danych testowych .....	164
Uruchamianie zadania w lokalnym mechanizmie wykonywania zadań .....	165
Testowanie sterownika .....	166
Uruchamianie programów w klastrze .....	167
Tworzenie pakietu z zadaniem .....	168
Uruchamianie zadania .....	169
Sieciowy interfejs użytkownika modelu MapReduce .....	171
Pobieranie wyników .....	174
Debugowanie zadania .....	175
Dzienniki w Hadoopie .....	178
Zdalne diagnozowanie .....	180
Dostrajanie zadania .....	181
Profilowanie operacji .....	181
Przepływ pracy w modelu MapReduce .....	182
Rozbijanie problemu na zadania w modelu MapReduce .....	183
JobControl .....	184
Apache Oozie .....	185

### **Rozdział 7. Jak działa model MapReduce? ..... 191**

Wykonywanie zadań w modelu MapReduce .....	191
Przesyłanie zadania .....	192
Inicjowanie zadania .....	193
Przypisywanie operacji do węzłów .....	194

Wykonywanie operacji .....	194
Aktualizowanie informacji o postępie i statusu .....	196
Ukończenie zadania .....	197
Niepowodzenia .....	198
Niepowodzenie operacji .....	198
Niepowodzenie zarządcy aplikacji .....	199
Niepowodzenie menedżera węzła .....	200
Niepowodzenie menedżera zasobów .....	201
Przestawianie i sortowanie .....	202
Etap mapowania .....	202
Etap redukcji .....	203
Dostrajanie konfiguracji .....	206
Wykonywanie operacji .....	208
Środowisko wykonywania operacji .....	208
Wykonywanie spekulacyjne .....	209
Klasy z rodziny OutputCommitter .....	210

## **Rozdział 8. Typy i formaty z modelu MapReduce ..... 213**

Typy w modelu MapReduce .....	213
Domyślne zadanie z modelu MapReduce .....	216
Formaty wejściowe .....	222
Wejściowe porcje danych i rekordy .....	222
Tekstowe dane wejściowe .....	232
Binarne dane wejściowe .....	236
Różne dane wejściowe .....	237
Dane wejściowe (i wyjściowe) z bazy .....	238
Formaty danych wyjściowych .....	238
Tekstowe dane wyjściowe .....	239
Binarne dane wyjściowe .....	239
Wiele danych wyjściowych .....	240
Leniwe generowanie danych wyjściowych .....	243
Dane wyjściowe dla bazy .....	244

## **Rozdział 9. Funkcje modelu MapReduce ..... 245**

Liczniki .....	245
Liczniki wbudowane .....	245
Zdefiniowane przez użytkowników liczniki Javy .....	248
Zdefiniowane przez użytkownika liczniki narzędzia Streaming .....	252
Sortowanie .....	253
Przygotowania .....	253
Częściowe sortowanie .....	254
Sortowanie wszystkich danych .....	255
Sortowanie pomocnicze .....	259
Złączanie .....	264
Złączanie po stronie mapowania .....	265
Złączanie po stronie redukcji .....	265
Rozdzielanie danych pomocniczych .....	268
Wykorzystanie konfiguracji zadania .....	268
Rozproszona pamięć podręczna .....	269
Klasy biblioteczne modelu MapReduce .....	273

## Część III. Praca z platformą Hadoop ..... 275

### Rozdział 10. Budowanie klastra opartego na platformie Hadoop ..... 277

Specyfikacja klastra .....	278
Określanie wielkości klastra .....	279
Topologia sieci .....	280
Przygotowywanie i instalowanie klastra .....	282
Instalowanie Javy .....	282
Tworzenie kont użytkowników w Uniksie .....	282
Instalowanie Hadoopa .....	282
Konfigurowanie ustawień protokołu SSH .....	282
Konfigurowanie Hadoopa .....	283
Formatowanie systemu plików HDFS .....	283
Uruchamianie i zatrzymywanie demonów .....	284
Tworzenie katalogów użytkowników .....	285
Konfiguracja Hadoopa .....	285
Zarządzanie konfiguracją .....	286
Ustawienia środowiskowe .....	287
Ważne właściwości demonów Hadoopa .....	289
Adresy i porty demonów Hadoopa .....	296
Inne właściwości Hadoopa .....	298
Bezpieczeństwo .....	299
Kerberos i Hadoop .....	300
Tokeny do delegowania uprawnień .....	302
Inne uprawnienia w zabezpieczeniach .....	303
Testy porównawcze klastra opartego na Hadoopie .....	305
Testy porównawcze w Hadoopie .....	305
Zadania użytkowników .....	307

### Rozdział 11. Zarządzanie platformą Hadoop ..... 309

System HDFS .....	309
Trwałe struktury danych .....	309
Tryb bezpieczny .....	314
Rejestrowanie dziennika inspekcji .....	315
Narzędzia .....	316
Monitorowanie .....	320
Rejestrowanie informacji w dziennikach .....	320
Wskaźniki i technologia JMX .....	321
Konserwacja .....	322
Standardowe procedury administracyjne .....	322
Dodawanie i usuwanie węzłów .....	324
Aktualizacje .....	327

<b>Część IV. Powiązane projekty .....</b>	<b>331</b>
<b>Rozdział 12. Avro .....</b>	<b>333</b>
Typy danych i schematy systemu Avro .....	334
Serializacja i deserializacja w pamięci .....	337
Specyficzny interfejs API .....	338
Pliki danych systemu Avro .....	340
Współdziałanie języków .....	341
Interfejs API dla Pythona .....	341
Narzędzia systemu Avro .....	342
Określanie schematu .....	343
Porządek sortowania .....	344
Avro i model MapReduce .....	346
Sortowanie za pomocą modelu MapReduce i systemu Avro .....	349
Używanie systemu Avro w innych językach .....	351
<b>Rozdział 13. Parquet .....</b>	<b>353</b>
Model danych .....	354
Kodowanie struktury zagnieżdżonych danych .....	355
Format plików Parquet .....	356
Konfiguracja dla formatu Parquet .....	358
Zapis i odczyt plików w formacie Parquet .....	358
Avro, Protocol Buffers i Thrift .....	360
Format Parquet i model MapReduce .....	362
<b>Rozdział 14. Flume .....</b>	<b>365</b>
Instalowanie platformy Flume .....	365
Przykład .....	366
Transakcje i niezawodność .....	368
Porcje zdarzeń .....	369
Ujścia w systemie HDFS .....	369
Podział na partycje i interceptory .....	370
Formaty plików .....	371
Rozsyłanie danych do wielu kanałów .....	372
Gwarancje dostarczenia .....	373
Selektory replikacji i rozsyłania .....	374
Dystrybucja — warstwy agentów .....	374
Gwarancje dostarczenia danych .....	376
Grupy ujść .....	377
Integrowanie platformy Flume z aplikacjami .....	380
Katalog komponentów .....	381
Dalsza lektura .....	382



<b>Rozdział 15. Sqoop .....</b>	<b>383</b>
Pobieranie Sqooa .....	383
Konektory Sqooa .....	385
Przykładowa operacja importu .....	385
Formaty plików tekstowych i binarnych .....	388
Wygenerowany kod .....	388
Inne systemy serializacji .....	389
Importowanie — dokładne omówienie .....	389
Kontrolowanie procesu importu .....	391
Import i spójność .....	392
Przyrostowy import .....	392
Importowanie w trybie bezpośrednim .....	392
Praca z zaimportowanymi danymi .....	393
Importowane dane i platforma Hive .....	394
Importowanie dużych obiektów .....	396
Eksportowanie .....	398
Eksportowanie — dokładne omówienie .....	399
Eksport i transakcje .....	401
Eksport i pliki typu SequenceFile .....	401
Dalsza lektura .....	402
<b>Rozdział 16. Pig .....</b>	<b>403</b>
Instalowanie i uruchamianie platformy Pig .....	404
Tryby wykonywania .....	404
Uruchamianie programów platformy Pig .....	406
Grunt .....	406
Edytory kodu w języku Pig Latin .....	407
Przykład .....	407
Generowanie przykładowych danych .....	409
Porównanie platformy Pig z bazami danych .....	410
Język Pig Latin .....	411
Struktura .....	411
Instrukcje .....	412
Wyrażenia .....	417
Typy .....	418
Schematy .....	419
Funkcje .....	423
Makra .....	425
Funkcje zdefiniowane przez użytkownika .....	426
Funkcje UDF służące do filtrowania .....	426
Obliczeniowa funkcja UDF .....	429
Funkcje UDF służące do wczytywania danych .....	430
Operatory używane do przetwarzania danych .....	433
Wczytywanie i zapisywanie danych .....	433
Filtrowanie danych .....	434
Grupowanie i złączanie danych .....	436
Sortowanie danych .....	441
Łączenie i dzielenie danych .....	442

Platforma Pig w praktyce .....	442
Współbieżność .....	442
Relacje anonimowe .....	443
Podstawianie wartości pod parametry .....	443
Dalsza lektura .....	444

## **Rozdział 17. Hive ..... 445**

Instalowanie platformy Hive .....	446
Powłoka platformy Hive .....	446
Przykład .....	448
Uruchamianie platformy Hive .....	449
Konfigurowanie platformy Hive .....	449
Usługi platformy Hive .....	451
Magazyn metadanych .....	453
Porównanie z tradycyjnymi bazami danych .....	456
Uwzględnianie schematu przy odczycie lub przy zapisie .....	456
Aktualizacje, transakcje i indeksy .....	456
Inne silniki obsługujące język SQL w Hadoopie .....	457
HiveQL .....	458
Typy danych .....	458
Operatory i funkcje .....	462
Tabele .....	463
Tabele zarządzane i tabele zewnętrzne .....	463
Partycje i kubelki .....	464
Formaty przechowywania danych .....	468
Importowanie danych .....	472
Modyfikowanie tabel .....	473
Usuwanie tabel .....	474
Pobieranie danych .....	474
Sortowanie i agregacja danych .....	475
Skrypty modelu MapReduce .....	475
Złączenia .....	476
Podzapytania .....	479
Widoki .....	480
Funkcje zdefiniowane przez użytkowników .....	481
Pisanie funkcji UDF .....	482
Pisanie funkcji UDAF .....	484
Dalsza lektura .....	488

## **Rozdział 18. Crunch ..... 489**

Przykład .....	490
Podstawowe interfejsy API Cruncha .....	493
Proste operacje .....	493
Typy .....	497
Źródłowe i docelowe zbiory danych .....	500
Funkcje .....	502
Materializacja .....	504

Wykonywanie potoku .....	506
Uruchamianie potoku .....	506
Zatrzymywanie potoku .....	507
Inspekcja planu wykonania w Crunchu .....	508
Algorytmy iteracyjne .....	511
Tworzenie punktów kontrolnych w potokach .....	512
Biblioteki w Crunchu .....	513
Dalsza lektura .....	515

## **Rozdział 19. Spark ..... 517**

Instalowanie Sparka .....	518
Przykład .....	518
Aplikacje, zadania, etapy i operacje w Sparku .....	520
Niezależna aplikacja w języku Scala .....	520
Przykład napisany w Javie .....	521
Przykład napisany w Pythonie .....	522
Zbiory RDD .....	523
Tworzenie zbiorów RDD .....	523
Transformacje i akcje .....	524
Utrwalanie danych .....	527
Serializacja .....	529
Zmienne współużytkowane .....	530
Zmienne rozsyłane .....	531
Akumulatory .....	531
Anatomia przebiegu zadania w Sparku .....	532
Przesyłanie zadań .....	532
Tworzenie skierowanego grafu acyklicznego .....	533
Szeregowanie operacji .....	535
Wykonywanie operacji .....	536
Wykonawcy i menedżery klastra .....	536
Spark i YARN .....	537
Dalsza lektura .....	540

## **Rozdział 20. HBase ..... 541**

Podstawy .....	541
Tło historyczne .....	542
Omówienie zagadnień .....	542
Krótki przegląd modelu danych .....	542
Implementacja .....	544
Instalacja .....	546
Przebieg testowy .....	547
Klienci .....	549
Java .....	549
Model MapReduce .....	552
Interfejsy REST i Thrift .....	553
Budowanie interaktywnej aplikacji do przesyłania zapytań .....	553
Projekt schematu .....	554
Wczytywanie danych .....	555
Zapytania interaktywne .....	558

Baza HBase a bazy RDBMS .....	561
Historia cieszącej się powodzeniem usługi .....	562
Baza HBase .....	563
Bazy HBase w praktyce .....	563
System HDFS .....	564
Interfejs użytkownika .....	564
Wskaźniki .....	565
Liczniki .....	565
Dalsza lektura .....	565

## **Rozdział 21. ZooKeeper .....567**

Instalowanie i uruchamianie systemu ZooKeeper .....	568
Przykład .....	570
Przynależność do grupy w systemie ZooKeeper .....	570
Tworzenie grupy .....	571
Dołączanie członków do grupy .....	573
Wyświetlanie członków grupy .....	574
Usuwanie grupy .....	575
Usługa ZooKeeper .....	576
Model danych .....	576
Operacje .....	578
Implementacja .....	582
Spójność .....	583
Sesje .....	585
Stany .....	587
Budowanie aplikacji z wykorzystaniem ZooKeepera .....	588
Usługa do zarządzania konfiguracją .....	588
Odporna na błędy aplikacja ZooKeepera .....	591
Usługa do zarządzania blokadami .....	594
Inne rozproszone struktury danych i protokoły .....	596
ZooKeeper w środowisku produkcyjnym .....	597
Odporność a wydajność .....	598
Konfiguracja .....	599
Dalsza lektura .....	600

## **Część V. Studia przypadków .....601**

### **Rozdział 22. Integrowanie danych w firmie Cerner .....603**

Od integracji procesorów do integracji semantycznej .....	603
Poznaj platformę Crunch .....	604
Budowanie kompletnego obrazu .....	604
Integrowanie danych z obszaru opieki zdrowotnej .....	607
Możliwość łączenia danych w różnych platformach .....	610
Co dalej? .....	611

<b>Rozdział 23. Nauka o danych biologicznych</b>	
<b>— ratowanie życia za pomocą oprogramowania .....</b>	<b>613</b>
Struktura DNA .....	615
Kod genetyczny — przekształcanie liter DNA w białka .....	616
Traktowanie kodu DNA jak kodu źródłowego .....	617
Projekt poznania ludzkiego genomu i genomu referencyjne .....	619
Sekwencjonowanie i wyrównywanie DNA .....	620
ADAM — skalowalna platforma do analizy genomu .....	621
Programowanie piśmienne za pomocą języka IDL systemu Avro .....	621
Dostęp do danych kolumnowych w formacie Parquet .....	623
Prosty przykład — zliczanie k-merów za pomocą Sparka i ADAM-a .....	624
Od spersonalizowanych reklam do spersonalizowanej medycyny .....	626
Dołącz do projektu .....	627
<b>Rozdział 24. Cascading .....</b>	<b>629</b>
Pola, krotki i potoki .....	630
Operacje .....	632
Typy Tap, Scheme i Flow .....	634
Cascading w praktyce .....	635
Elastyczność .....	637
Hadoop i Cascading w serwisie ShareThis .....	638
Podsumowanie .....	642
<b>Dodatki .....</b>	<b>643</b>
<b>Dodatek A. Instalowanie platformy Apache Hadoop .....</b>	<b>645</b>
Wymagania wstępne .....	645
Instalacja .....	645
Konfiguracja .....	646
Tryb niezależny .....	647
Tryb pseudorozproszony .....	647
Tryb rozproszony .....	649
<b>Dodatek B. Dystrybucja firmy Cloudera .....</b>	<b>651</b>
<b>Dodatek C. Przygotowywanie danych meteorologicznych od instytucji NCDC .....</b>	<b>653</b>
<b>Dodatek D. Dawny i nowy interfejs API Javy dla modelu MapReduce .....</b>	<b>657</b>
<b>Skorowidz .....</b>	<b>661</b>



# Budowanie aplikacji w modelu MapReduce

W rozdziale 2. znajduje się wprowadzenie do modelu MapReduce. W tym rozdziale zapoznasz się z praktycznymi aspektami budowania w Hadoopie aplikacji w modelu MapReduce.

Pisanie programu w modelu MapReduce odbywa się zgodnie z określonym wzorcem. Najpierw należy przygotować funkcje mapującą i redukującą (najlepiej z testami jednostkowymi sprawdzającymi, czy funkcje działają zgodnie z oczekiwaniami). Następnie trzeba dodać program sterujący, który uruchamia zadanie. Ten program można wywołać w środowisku IDE dla małego podzbioru danych, aby ustalić, czy działa poprawnie. Jeśli wystąpią problemy, należy wykorzystać debugger ze środowiska IDE do wykrycia ich źródła. Na podstawie uzyskanych informacji można rozwinąć testy jednostkowe z uwzględnieniem danej sytuacji i usprawnić mapper lub reduktor, by zapewnić poprawną obsługę używanych danych wejściowych.

Jeżeli program działa prawidłowo dla małego zbioru danych, jest gotowy do uruchomienia w klastrze. Wykonanie programu dla pełnego zbioru danych prawdopodobnie pozwoli odkryć inne problemy. Można je rozwiązać w opisany wcześniej sposób, rozbudowując testy i modyfikując mapper lub reduktor pod kątem obsługi nowych przypadków. Debugowanie błędnych programów w klastrze jest trudne, dlatego dalej opisanych jest kilka popularnych technik, które to ułatwiają.

Gdy program już działa, możesz go dopracować. Najpierw wykonaj standardowe testy związane z przyspieszaniem pracy programów w modelu MapReduce, a następnie przeprowadź profilowanie zadania. Profilowanie programów rozproszonych to skomplikowane zadanie, jednak Hadoop udostępni mechanizmy ułatwiające ten proces.

Przed rozpoczęciem pisania programu w modelu MapReduce należy zainstalować i skonfigurować środowisko programowania. W tym celu musisz zrozumieć, jak Hadoop przetwarza konfigurację.

## API do obsługi konfiguracji

Komponenty w Hadoopie są konfigurowane za pomocą dostępnego w tej platformie specjalnego interfejsu API. Obiekt klasy `Configuration` (z pakietu `org.apache.hadoop.conf`) reprezentuje kolekcję *właściwości* konfiguracyjnych i ich wartości. Każda właściwość ma nazwę (typu `String`), a jako wartości używane są różne typy, w tym typy proste Javy (na przykład `boolean`, `int`, `long` i `float`), typy `String`, `Class` i `java.io.File` oraz kolekcje wartości typu `String`.

Obiekt typu `Configuration` wczytuje właściwości z zasobów — plików XML o prostej strukturze definiujących pary nazwa-wartość (zobacz listing 6.1).

Listing 6.1. Prosty plik konfiguracyjny *configuration-1.xml*

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>color</name>
    <value>yellow</value>
    <description>Color</description>
  </property>

  <property>
    <name>size</name>
    <value>10</value>
    <description>Size</description>
  </property>

  <property>
    <name>weight</name>
    <value>heavy</value>
    <final>true</final>
    <description>Weight</description>
  </property>

  <property>
    <name>size-weight</name>
    <value>${size},${weight}</value>
    <description>Size and weight</description>
  </property>
</configuration>
```

Jeśli dane obiektu typu `Configuration` znajdują się w pliku *configuration-1.xml*, dostęp do właściwości można uzyskać za pomocą następującego kodu:

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
assertThat(conf.get("color"), is("yellow"));
assertThat(conf.getInt("size", 0), is(10));
assertThat(conf.get("breadth", "wide"), is("wide"));
```

Warto zwrócić uwagę na kilka kwestii. W pokazanym pliku XML nie są przechowywane informacje o typie. Właściwości można interpretować jako wartość danego typu w momencie ich wczytywania. Ponadto metody `get()` umożliwiają określenie wartości domyślnej, używanej, jeśli dana właściwość nie jest zdefiniowana w pliku XML (tak jest z właściwością `breadth` w przykładzie).

## Łączenie zasobów

Sytuacja staje się ciekawa, gdy do definiowania obiektu typu `Configuration` używany jest więcej niż jeden zasób. To podejście jest stosowane w Hadoopie do wyodrębniania właściwości domyślnych systemu (zdefiniowanych wewnętrznie w pliku *core-default.xml*) od właściwości specyficznych dla danej jednostki (zdefiniowanych w pliku *core-site.xml*). W pliku z listingu 6.2 zdefiniowane są właściwości `size` i `weight`.



Listing 6.2. Drugi plik konfiguracyjny — *configuration-2.xml*

```
<?xml version="1.0"?>
<configuration>
  <property>
    <name>size</name>
    <value>12</value>
  </property>

  <property>
    <name>weight</name>
    <value>light</value>
  </property>
</configuration>
```

Zasoby są dodawane do obiektu typu `Configuration` po kolei:

```
Configuration conf = new Configuration();
conf.addResource("configuration-1.xml");
conf.addResource("configuration-2.xml");
```

Właściwości zdefiniowane w później dodawanych zasobach zastępują wcześniejsze definicje. Dlatego właściwość `size` ostatecznie przyjmuje wartość z drugiego pliku konfiguracyjnego, *configuration-2.xml*.

```
assertThat(conf.getInt("size", 0), is(12));
```

Jednak właściwości oznaczonych jako `final` nie można zastępować w dalszych definicjach. Właściwość `weight` w pierwszym pliku konfiguracyjnym ma modyfikator `final`, dlatego próba zastąpienia jej wartości w drugim pliku kończy się niepowodzeniem (zachowana zostaje wartość z pierwszego pliku).

```
assertThat(conf.get("weight"), is("heavy"));
```

Próba przesłonięcia właściwości z modyfikatorem `final` zwykle oznacza błąd konfiguracji. Skutkuje to zarejestrowaniem komunikatu ostrzegawczego, co ułatwia późniejszą diagnozę. Administratorzy oznaczają modyfikatorem `final` właściwości w lokalnych plikach demonów, gdy nie chcą, aby użytkownicy zmieniali te właściwości w plikach po stronie klienta lub za pomocą parametrów przekazywanych do zadania.

## Podstawianie wartości zmiennych

Właściwości konfiguracyjne można definiować za pomocą innych właściwości i właściwości systemowych. Na przykład właściwość `size-weight` z pierwszego pliku konfiguracyjnego jest zdefiniowana jako `${size},${weight}`. Za te zmienne podstawiane są wartości znalezione w konfiguracji.

```
assertThat(conf.get("size-weight"), is("12,heavy"));
```

Właściwości systemowe są traktowane priorytetowo względem właściwości zdefiniowanych w plikach zasobów.

```
System.setProperty("size", "14");
assertThat(conf.get("size-weight"), is("14,heavy"));
```

Ten mechanizm jest przydatny do zastępowania właściwości w wierszu poleceń za pomocą argumentów maszyny JVM `-Dwłaścivość=wartość`.

Zauważ, że choć właściwości konfiguracyjne można definiować w kategoriach właściwości systemowych, to jeśli właściwości systemowe nie zostaną zredefiniowane we właściwościach konfiguracyjnych, *nie* będą dostępne za pomocą interfejsu API do obsługi konfiguracji. Dlatego:

```
System.setProperty("length", "2");
assertThat(conf.get("length"), is((String) null));
```

## Przygotowywanie środowiska programowania

Pierwszy krok polega na utworzeniu projektu. Pozwoli to budować programy w modelu MapReduce i uruchamiać je w trybie lokalnym (niezależnym) z poziomu wiersza poleceń lub środowiska IDE. Plik POM (ang. *Project Object Model*) Mavena z listingu 6.3 pokazuje zależności potrzebne do budowania i testowania programów w modelu MapReduce.

*Listing 6.3. Plik POM Mavena potrzebny do budowania i testowania aplikacji w modelu MapReduce*

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.hadoopbook</groupId>
  <artifactId>hadoop-book-mr-dev</artifactId>
  <version>4.0</version>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <hadoop.version>2.5.1</hadoop.version>
  </properties>
  <dependencies>
    <!-- Element głównego klienta Hadoopa -->
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-client</artifactId>
      <version>${hadoop.version}</version>
    </dependency>
    <!-- Elementy testów jednostkowych -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.apache.mrunit</groupId>
      <artifactId>mrunit</artifactId>
      <version>1.1.0</version>
      <classifier>hadoop2</classifier>
      <scope>test</scope>
    </dependency>
    <!-- Element testowy Hadoopa do uruchamiania miniklastrów -->
    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-minicluster</artifactId>
      <version>${hadoop.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <build>
    <finalName>hadoop-examples</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
```

```

    </configuration>
  </plugin>
</plugins>
</build>
</project>

```

Interesującą częścią pliku POM jest sekcja z zależnościami (możesz też zastosować inne narzędzie do budowania, takie jak Gradle lub Ant z Ivy, o ile użyjesz zdefiniowanego tu zestawu zależności). Do budowania zadań w modelu MapReduce potrzebny jest tylko element `hadoop-client`, obejmujący wszystkie używane po stronie klienta klasy Hadoopa potrzebne do interakcji z systemem HDFS i modelem MapReduce. Do przeprowadzania testów jednostkowych używany jest tu `JUnit`, a do pisania testów dla modelu MapReduce — `mrunit`. Biblioteka `hadoop-minicluster` obsługuje miniklastry przydatne do testowania programów na klastrach Hadoopa działających w jednej maszynie JVM.

Wiele środowisk IDE potrafi bezpośrednio wczytywać pliki POM Mavena, dlatego wystarczy wskazać im katalog z plikiem `pom.xml`, po czym można przejść do pisania kodu. Możesz też wykorzystać Mavena do wygenerowania plików konfiguracyjnych dla środowiska IDE. Na przykład poniższy kod tworzy pliki konfiguracyjne dla środowiska Eclipse, co pozwala zaimportować projekt do tego narzędzia.

```
% mvn eclipse:eclipse -DdownloadSources=true -DdownloadJavadocs=true
```

## Zarządzanie konfiguracją

W trakcie budowania aplikacji Hadoopa programista często przełącza aplikację między środowiskiem lokalnym a klastrzem. Możliwe też, że pracuje z kilkoma klastrami lub używa lokalnego „pseudorozproszzonego” klastra, w którym chce przeprowadzać testy (w takim klastrze wszystkie demony działają na jednej maszynie; ustawianie tego trybu opisano w dodatku A).

Jednym ze sposobów na uwzględnienie wszystkich tych sytuacji jest utworzenie plików konfiguracyjnych Hadoopa z ustawieniami połączeń dla każdego używanego klastra. Następnie w momencie uruchamiania aplikacji lub narzędzi Hadoopa należy wskazać potrzebny klaster. Zaleca się przechowywanie wspomnianych plików poza drzewem katalogów instalacji Hadoopa, ponieważ pozwala to na łatwe przełączanie się między wersjami Hadoopa bez powielania lub utraty ustawień.

Na potrzeby książki założmy, że istnieje katalog `conf` zawierający trzy pliki konfiguracyjne: `hadoop-local.xml`, `hadoop-localhost.xml` i `hadoop-cluster.xml` (są one dostępne w przykładowym kodzie dołączonym do książki). Zauważ, że w nazwach tych plików nie ma nic specjalnego. Te pliki to tylko wygodny sposób na połączenie wybranych ustawień konfiguracyjnych. Możesz porównać je z plikami z tabeli A.1 z dodatku A, w których umieszczane są analogiczne ustawienia konfiguracyjne po stronie serwera.

Plik *hadoop-local.xml* zawiera domyślną konfigurację Hadoopa dla domyślnego systemu plików i lokalnej (z maszyny JVM) platformy do uruchamiania zadań w modelu MapReduce.

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.defaultFS</name>
    <value>file:///</value>
  </property>

  <property>
    <name>mapreduce.framework.name</name>
    <value>local</value>
  </property>

</configuration>
```

Ustawienia z pliku *hadoop-localhost.xml* określają węzeł nazw i menedżer zasobów systemu YARN działające w hoście lokalnym.

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost/</value>
  </property>

  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>

  <property>
    <name>yarn.resourcemanager.address</name>
    <value>localhost:8032</value>
  </property>

</configuration>
```

Plik *hadoo-cluster.xml* zawiera szczegółowe informacje o adresach węzła nazw i menedżera zasobów systemu YARN z klastra. W praktyce ten plik jest nazywany na podstawie nazwy klastra, a nie, tak jak tutaj, „cluster”.

```
<?xml version="1.0"?>
<configuration>

  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://namenode/</value>
  </property>

  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>

  <property>
    <name>yarn.resourcemanager.address</name>
    <value>resourcemanager:8032</value>
  </property>

</configuration>
```

W razie potrzeby możesz dodać do tych plików inne właściwości konfiguracyjne.

## Ustawianie tożsamości użytkownika

Tożsamość użytkownika używana w Hadoopie do zarządzania uprawnieniami w systemie HDFS jest ustalana za pomocą polecenia `whoami` wywoływanego w systemie klienta. Podobnie nazwy grup są określone na podstawie danych wyjściowych z instrukcji `groups`.

Jeśli jednak tożsamość użytkownika w Hadoopie jest różna od nazwy konta użytkownika na maszynie klienckiej, można bezpośrednio ustawić nazwę użytkownika w Hadoopie za pomocą zmiennej środowiskowej `HADOOP_USER_NAME`. Możesz też zmienić odwzorowania w grupach użytkowników za pomocą właściwości konfiguracyjnej `hadoop.user.group.static.mapping.overrides`. Na przykład wartość `dr.who=;preston=directors, inventors` oznacza, że użytkownik `dr.who` nie jest członkiem żadnej grupy, a użytkownik `preston` należy do grup `directors` i `inventors`.

Tożsamość użytkownika używaną przez interfejsy sieciowe Hadoopa można ustawić za pomocą właściwości `hadoop.http.staticuser.user`. Domyślnie używana jest tożsamość `dr.who`, która nie daje uprawnień administratora, dlatego pliki systemowe nie są dostępne z poziomu interfejsów sieciowych.

Zauważ, że domyślnie w tym modelu uwierzytelnianie nie jest stosowane. Z punktu „Inne uprawnienia w zabezpieczeniach” w rozdziale 10. dowiesz się, jak skonfigurować w Hadoopie uwierzytelnianie oparte na protokole Kerberos.

Przy tych ustawieniach łatwo można wskazać dowolną konfigurację za pomocą opcji `-conf` w wierszu poleceń. Na przykład poniższe polecenie wyświetla zawartość katalogu z serwera HDFS działającego w trybie pseudorozproszonym na hoście lokalnym.

```
% hadoop fs -conf conf/hadoop-localhost.xml -ls .
Found 2 items
drwxr-xr-x - tom supergroup      0 2014-09-08 10:19 input
drwxr-xr-x - tom supergroup      0 2014-09-08 10:19 output
```

Jeśli pominiesz opcję `-conf`, użyta zostanie konfiguracja Hadoopa z podkatalogu `etc/hadoop` z katalogu ze zmiennej `$HADOOP_HOME`. Jeżeli jednak ustawiona jest zmienna `HADOOP_CONF_DIR`, pliki konfiguracyjne Hadoopa są wczytywane z podanej w niej lokalizacji.



Oto inny sposób zarządzania ustawieniami konfiguracyjnymi — skopiuj katalog `etc/hadoop` z instalacji Hadoopa w inne miejsce, umieść tam pliki konfiguracyjne `*-site.xml` (z odpowiednimi opcjami) i ustaw zmienną środowiskową `HADOOP_CONF_DIR` na tę nową lokalizację. Główna zaleta takiego podejścia polega na tym, że nie trzeba używać opcji `-conf` w każdej instrukcji. Ponadto można wprowadzić zmiany w plikach innych niż pliki XML z konfiguracją Hadoopa (na przykład w pliku `log4j.properties`), ponieważ katalog ze zmiennej `HADOOP_CONF_DIR` zawiera kopię wszystkich plików konfiguracyjnych (zobacz punkt „Konfiguracja Hadoopa” w rozdziale 10.).

Dostępne w Hadoopie narzędzia obsługują opcję `-conf`. Można też łatwo zapewnić jej obsługę we własnych programach (na przykład w programach uruchamiających zadania w modelu MapReduce). Umożliwia to interfejs `Tool`.

## GenericOptionsParser, Tool i ToolRunner

Hadoop udostępnia kilka klas pomocniczych ułatwiających uruchamianie zadań z poziomu wiersza poleceń. `GenericOptionsParser` to klasa, która interpretuje standardowe opcje Hadoopa podawane w wierszu poleceń i ustawia je w obiekcie typu `Configuration`, tak by można je było potem w dowolny sposób zastosować. Programiści zwykle nie używają bezpośrednio klasy `GenericOptionsParser`, ponieważ wygodniej jest zaimplementować interfejs `Tool` i uruchamiać aplikację za pomocą klasy `ToolRunner`, która wewnętrznie korzysta z klasy `GenericOptionsParser`.

```
public interface Tool extends Configurable {
    int run(String [] args) throws Exception;
}
```

Listing 6.4 przedstawia bardzo prostą implementację interfejsu `Tool`, która wyświetla klucze i wartości wszystkich właściwości z obiektu typu `Configuration`.

*Listing 6.4. Przykładowa implementacja interfejsu `Tool` wyświetlająca właściwości z obiektu typu `Configuration`*

```
public class ConfigurationPrinter extends Configured implements Tool {

    static {
        Configuration.addDefaultResource("hdfs-default.xml");
        Configuration.addDefaultResource("hdfs-site.xml");
        Configuration.addDefaultResource("yarn-default.xml");
        Configuration.addDefaultResource("yarn-site.xml");
        Configuration.addDefaultResource("mapred-default.xml");
        Configuration.addDefaultResource("mapred-site.xml");
    }

    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = getConf();
        for (Entry<String, String> entry: conf) {
            System.out.printf("%s=%s\n", entry.getKey(), entry.getValue());
        }
        return 0;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new ConfigurationPrinter(), args);
        System.exit(exitCode);
    }
}
```

`ConfigurationPrinter` to klasa pochodna od `Configured` (która implementuje interfejs `Configurable`). Wszystkie implementacje interfejsu `Tool` muszą też implementować interfejs `Configurable`, ponieważ `Tool` to interfejs pochodny od `Configurable`. Utworzenie klasy pochodnej od `Configured` to często najłatwiejszy sposób na zrealizowanie tego wymogu. Metoda `run()` pobiera obiekt typu `Configuration` za pomocą metody `getConf()` z interfejsu `Configurable`, a następnie przechodzi po tym obiekcie i wyświetla każdą właściwość w standardowym wyjściu.

Statyczny blok gwarantuje, że oprócz podstawowych ustawień (znanych już obiektowi klasy `Configuration`) pobrane zostaną konfiguracje systemów HDFS i YARN oraz modelu MapReduce.

Metoda `main()` klasy `ConfigurationPrinter` nie wywołuje bezpośrednio własnej metody `run()`. Zamiast tego uruchamia statyczną metodę `run()` klasy `ToolRunner`, która przed wywołaniem tej metody tworzy obiekt typu `Configuration` na potrzeby interfejsu `Tool`. Klasa `ToolRunner` używa klasy

GenericOptionsParser do pobierania standardowych opcji z wiersza poleceń i ustawiania ich w obiekcie klasy Configuration. Efekt pobrania właściwości podanych w pliku *conf/hadoop-localhost.xml* można zobaczyć dzięki uruchomieniu poniższych instrukcji.

```
% mvn compile
% export HADOOP_CLASSPATH=target/classes/
% hadoop ConfigurationPrinter -conf conf/hadoop-localhost.xml \
  | grep yarn.resourcemanager.address=
yarn.resourcemanager.address=localhost:8032
```

## Które właściwości programista może ustawić?

Klasa ConfigurationPrinter to przydatne narzędzie do wykrywania, która właściwość jest ustawiona w środowisku. Konfigurację działającego demona (na przykład węzła nazw) można zobaczyć na stronie /conf serwera WWW. Numery portów znajdziesz w tabeli 10.6.

Aby zobaczyć domyślne ustawienia wszystkich publicznych właściwości Hadoopa, otwórz katalog *share/doc* w instalacji Hadoopa i poszukaj plików *core-default.xml*, *hdfs-default.xml*, *yarn-default.xml* i *mapred-default.xml*. Każda właściwość ma opis wyjaśniający, do czego służy i jakie wartości przyjmuje.

Dokumentację plików z ustawieniami domyślnymi znajdziesz w internecie na stronach podanych pod adresem <http://hadoop.apache.org/docs/current/> (poszukaj opcji *Configuration* w nawigacji). By uzyskać szczegółowe informacje o konkretnej wersji Hadoopa, zastąp słowo *current* w podanym adresie URL członem *r<wersja>*, na przykład <http://hadoop.apache.org/docs/r2.5.2/>.

Zauważ, że niektóre właściwości nie mają znaczenia, gdy są ustawione w konfiguracji klienta. Na przykład jeśli przy przesyłaniu zadania ustawisz właściwość *yarn.nodemanager.resource.memory-mb* w oczekiwaniu, że zmieni to ilość pamięci dostępnej menedżerom węzłów wykonujących zadanie, zawiędziesz się, ponieważ ta właściwość jest uwzględniana tylko wtedy, jeżeli ustawiono ją w pliku *yarn-site.xml* menedżera węzła. Zwykle można określić docelowy komponent właściwości na podstawie jej nazwy. Ponieważ nazwa *yarn.nodemanager.resource.memory-mb* rozpoczyna się od członu *yarn.nodemanager*, można się domyślić, że tę właściwość można ustawić tylko dla demona menedżera węzła. Jednak ta reguła nie zawsze się sprawdza, dlatego czasem będziesz musiał posłużyć się metodą prób i błędów lub nawet zajrzeć do kodu źródłowego.

Od wersji Hadoop 2 zmieniono nazwy właściwości konfiguracyjnych na bardziej jednolite. Na przykład właściwości systemu HDFS związane z węzłem nazw mają obecnie przedrostek *dfs.namenode*, dlatego dawna właściwość *dfs.name.dir* to teraz *dfs.namenode.name.dir*. Podobnie właściwości dotyczące modelu MapReduce otrzymały przedrostek *mapreduce* zamiast dawnego przedrostka *mapred*. Dlatego poprzednia nazwa *mapred.job.name* to obecnie *mapreduce.job.name*.

W tej książce używane są nowe nazwy właściwości, co pozwala uniknąć ostrzeżeń o stosowaniu przestarzałych nazw. Dawne nazwy właściwości są jednak wciąż dozwolone i często pojawiają się w starszej dokumentacji. Listę przestarzałych nazw właściwości i ich nowych odpowiedników znajdziesz w witrynie Hadoopa (<http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/DeprecatedProperties.html>).

W tej książce opisano wiele spośród najważniejszych właściwości konfiguracyjnych Hadoopa.

Klasa GenericOptionsParser umożliwia też ustawianie pojedynczych właściwości. Oto przykład:

```
% hadoop ConfigurationPrinter -D color=yellow | grep color
color=yellow
```

Tu opcja `-D` jest używana do ustawienia właściwości konfiguracyjnej o kluczu `color` na wartość `yellow`. Ustawienia podane za pomocą opcji `-D` są traktowane priorytetowo względem właściwości z plików konfiguracyjnych. Jest to bardzo przydatne, ponieważ można zapisać w plikach konfiguracyjnych wartości domyślne, a następnie zastępować je za pomocą opcji `-D`. Przykładem jest ustawianie liczby reduktorów dla zadania w modelu MapReduce za pomocą opcji `-D mapreduce.job.reduces=n`. Powoduje to zmianę liczby reduktorów ustawionej w klastrze lub w plikach konfiguracyjnych po stronie klienta.

Inne opcje obsługiwane przez klasy `GenericOptionsParser` i `ToolRunner` są wymienione w tabeli 6.1. Więcej informacji o konfiguracyjnym interfejsie API Hadoopa zawiera punkt „API do obsługi konfiguracji”.

Tabela 6.1. Opcje dla klas `GenericOptionsParser` i `ToolRunner`

Opcja	Opis
<code>-D właściwość=wartość</code>	Ustawia właściwość konfiguracyjną Hadoopa na podaną wartość. Zastępuje konfiguracyjne właściwości domyślne i właściwości witryny, a także właściwości ustawione za pomocą opcji <code>-conf</code> .
<code>-conf nazwa_pliku ...</code>	Dodaje wskazane pliki do listy zasobów konfiguracyjnych. Jest to wygodny sposób ustawiania właściwości na poziomie witryny lub jednoczesnego określania wielu właściwości.
<code>-fs uri</code>	Ustawia domyślny system plików na podany identyfikator URI. To skrótowy zapis instrukcji <code>-D fs.defaultFS=uri</code> .
<code>-jt host:port</code>	Ustawia menedżera zasobów systemu YARN na określone hosta i port. W Hadoopie 1 instrukcja ustawia adres jobtrackera, stąd jej nazwa. To skrótowy zapis polecenia <code>-D yarn.resourcemanager.address=host:port</code> .
<code>-files plik1,plik2,...</code>	Kopiuje wskazane pliki z lokalnego systemu plików (lub dowolnego systemu plików, jeśli określony jest schemat) do współużytkowanego systemu plików, z którego korzysta platforma MapReduce (zwykle jest to system HDFS). Te pliki są udostępniane programom w modelu MapReduce w katalogu roboczym zadania. Więcej o mechanizmie rozproszonej pamięci podręcznej w kontekście kopiowania plików do maszyn z klastra dowiesz się z punktu „Rozproszona pamięć podręczna” w rozdziale 9.
<code>-archives archiwum1,archiwum2,...</code>	Kopiuje wskazane archiwa z lokalnego systemu plików (lub dowolnego systemu plików, jeśli określony jest schemat) do współużytkowanego systemu plików, z którego korzysta platforma MapReduce (zwykle jest to system HDFS), wypakowuje je i udostępnia programom w modelu MapReduce w katalogu roboczym zadania.
<code>-libjars jar1,jar2,...</code>	Kopiuje wskazane pliki JAR z lokalnego systemu plików (lub dowolnego systemu plików, jeśli określony jest schemat) do współużytkowanego systemu plików, z którego korzysta platforma MapReduce (zwykle jest to system HDFS), i dodaje je do ścieżki do klas zadania w modelu MapReduce. Ta opcja to przydatny sposób przesyłania plików JAR potrzebnych zadaniu.



Nie pomył ustawiania właściwości Hadoopa za pomocą opcji `-D właściwość=wartość` na potrzeby klas `GenericOptionsParser` i `ToolRunner` z ustawianiem właściwości systemowych maszyny JVM przy użyciu opcji `-Dwłaściwość=wartość` w instrukcji `java`. Składnia właściwości systemowych maszyny JVM nie zezwala na używanie spacji między literą `D` a nazwą właściwości, natomiast przy podawaniu właściwości dla klasy `GenericOptionsParser` spacja jest dopuszczalna.

Właściwości systemowe maszyny JVM są pobierane z klasy `java.lang.System`, przy czym właściwości Hadoopa są dostępne tylko za pomocą obiektu typu `Configuration`. Dlatego choć właściwość systemowa `color` została ustawiona (za pomocą zmiennej



HADOOP\_OPTS), poniższa instrukcja nic nie wyświetla, ponieważ klasa Configuration ↪Printer nie używa klasy System.

```
% HADOOP_OPTS='-Dcolor=yellow' \  
hadoop ConfigurationPrinter | grep color
```

Jeśli chcesz mieć możliwość ustawiania konfiguracji za pomocą właściwości systemowych, musisz umieścić potrzebne z nich w pliku konfiguracyjnym. Więcej dowiesz się z punktu „Podstawianie wartości zmiennych”.

## Pisanie testów jednostkowych za pomocą biblioteki MRUnit

Funkcje mapująca i redukująca z modelu MapReduce są łatwe do testowania w izolacji. Wynika to z natury funkcji. MRUnit (<https://mrunit.apache.org/>) to biblioteka do obsługi testów, która ułatwia przekazywanie znanych danych wejściowych do mappera lub reduktora i sprawdzanie, czy dane wyjściowe są zgodne z oczekiwaniami. Biblioteka MRUnit jest używana razem ze standardowymi platformami testowymi (takimi jak JUnit), dlatego można przeprowadzać testy zadań w modelu MapReduce w środowisku programowania. Na przykład wszystkie testy opisane w tym miejscu można uruchomić w środowisku IDE na podstawie instrukcji z punktu „Przygotowywanie środowiska programowania”.

### Mapper

Test mappera jest przedstawiony na listingu 6.5.

Listing 6.5. Test jednostkowy klasy MaxTemperatureMapper

```
import java.io.IOException;  
import org.apache.hadoop.io.*;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.junit.*;  
  
public class MaxTemperatureMapperTest {  
  
    @Test  
    public void processesValidRecord() throws IOException, InterruptedException {  
        Text value = new Text("0043011990999991950051518004+68750+023550FM-12+0382" +  
            // Rok ~~~~  
            "99999V0203201N00261220001CN9999999N9-00111+9999999999");  
            // Temperatura ~~~~~  
        new Mapper<LongWritable, Text, Text, IntWritable>()  
            .withMapper(new MaxTemperatureMapper())  
            .withInput(new LongWritable(0), value)  
            .withOutput(new Text("1950"), new IntWritable(-11))  
            .runTest();  
    }  
}
```

Ten test jest oparty na bardzo prostym pomysł — należy przekazać rekord z danymi o pogodzie jako dane wejściowe dla mappera i sprawdzić, czy dane wyjściowe to rok i temperatura.

Ponieważ testowany jest mapper, używana jest klasa Mapper z biblioteki MRUnit. Do tej klasy trzeba przekazać testowany mapper (MaxTemperatureMapper), wejściowe klucze i wartości, a także oczekiwane wyjściowe klucze (obiekt typu Text reprezentujący rok 1950) i wartości (obiekt typu

IntWritable reprezentujący temperaturę  $-1,1^{\circ}\text{C}$ ). Na koniec można wywołać metodę `runTest()`, aby przeprowadzić test. Jeśli mapper nie zwraca oczekiwanych danych wyjściowych, biblioteka MRUnit zgłasza niepowodzenie testu. Zauważ, że klucz wejściowy można ustawić na dowolną wartość, ponieważ przedstawiony mapper go ignoruje.

W sposób typowy dla programowania sterowanego testami można następnie utworzyć wersję klasy Mapper, która przejdzie test (zobacz listing 6.6). Ponieważ pokazywane klasy są w tym rozdziale stopniowo modyfikowane, każda jest umieszczona w innym pakiecie określającym jej wersję, co ułatwia prezentację kodu. Na przykład `v1.MaxTemperatureMapper` to pierwsza wersja klasy `MaxTemperatureMapper`. Oczywiście w praktyce przy modyfikowaniu klas nie umieszcza się ich w odrębnych pakietach.

Listing 6.6. Pierwsza wersja mappera przechodząca test `MaxTemperatureMapperTest`

```
public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        String line = value.toString();
        String year = line.substring(15, 19);
        int airTemperature = Integer.parseInt(line.substring(87, 92));
        context.write(new Text(year), new IntWritable(airTemperature));
    }
}
```

To bardzo prosta implementacja, która pobiera pola z rokiem i temperaturą z wiersza i zapisuje je w obiekcie typu `Context`. Dodajmy teraz wykrywanie brakujących wartości, które w surowych danych są reprezentowane jako temperatura `+9999`.

```
@Test
public void ignoresMissingTemperatureRecord() throws IOException,
    InterruptedException {
    Text value = new Text("00430119909999991950051518004+68750+023550FM-12+0382" +
        // Rok ~~~~
        "99999V0203201N00261220001CN99999999N9+99991+99999999999");
        // Temperatura ~~~~
    new MapDriver<LongWritable, Text, Text, IntWritable>()
        .withMapper(new MaxTemperatureMapper())
        .withInput(new LongWritable(0), value)
        .runTest();
}
```

Klasę `MapDriver` można wykorzystać do wykrywania zera, jednego lub większej liczby rekordów wyjściowych — zgodnie z liczbą wywołań metody `withOutput()`. W tej aplikacji rekordy, w których brakuje temperatury, są pomijane, dlatego test sprawdza, czy dla konkretnej wartości wejściowej rzeczywiście nie są zwracane żadne dane wyjściowe.

Nowy test kończy się niepowodzeniem, ponieważ wartość `+9999` nie jest traktowana jak specjalny przypadek. Zamiast dodawać kolejne linie kodu do mappera, można utworzyć odrębną klasę parsera i umieścić w niej kod odpowiedzialny za parsowanie, co przedstawia listing 6.7.

Listing 6.7. Klasa do parsowania rekordów z danymi o pogodzie w formacie organizacji NCDC

```
public class NcdcRecordParser {

    private static final int MISSING_TEMPERATURE = 9999;

    private String year;
    private int airTemperature;
    private String quality;

    public void parse(String record) {
        year = record.substring(15, 19);
        String airTemperatureString;
        // Usuwanie początkowego znaku plus, ponieważ metoda parseInt źle sobie z nim radzi (w wersjach sprzed Javy 7)
        if (record.charAt(87) == '+') {
            airTemperatureString = record.substring(88, 92);
        } else {
            airTemperatureString = record.substring(87, 92);
        }
        airTemperature = Integer.parseInt(airTemperatureString);
        quality = record.substring(92, 93);
    }

    public void parse(Text record) {
        parse(record.toString());
    }

    public boolean isValidTemperature() {
        return airTemperature != MISSING_TEMPERATURE && quality.matches("[01459]");
    }

    public String getYear() {
        return year;
    }

    public int getAirTemperature() {
        return airTemperature;
    }
}
```

Wynikowy mapper (wersja 2.) jest znacznie prostszy (zobacz listing 6.8). Wywołuje jedynie metodę `parse()` parsera (która parsuje potrzebne pola z wiersza danych wejściowych), za pomocą metody `isValidTemperature()` sprawdza, czy znaleziono prawidłową temperaturę, i jeśli tak się stało, pobiera rok i temperaturę za pomocą getterów parsera. Zauważ, że za sprawdzanie wartości pola stanu i wykrywanie braku temperatury odpowiada metoda `isValidTemperature()`, pozwalająca odfiltrować nieprawidłowe odczyty temperatury.



Inną korzyścią wynikającą z utworzenia klasy parsera jest to, że pozwala ona na łatwe napisanie powiązanych mapperów dla podobnych zadań bez konieczności duplikowania kodu. Ponadto można pisać testy jednostkowe bezpośrednio dla parsera, dzięki czemu są one bardziej precyzyjne.

Listing 6.8. Mapper używający klasy narzędziowej do parsowania rekordów

```
public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    private NcdcRecordParser parser = new NcdcRecordParser();
```

```

@Override
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

    parser.parse(value);
    if (parser.isValidTemperature()) {
        context.write(new Text(parser.getYear()),
            new IntWritable(parser.getAirTemperature()));
    }
}
}

```

Testy mappera kończą się już powodzeniem, można więc przejść do pisania reduktora.

## Reduktor

Reduktor ma znajdować maksymalną wartość dla danego klucza. Oto prosty test tego mechanizmu. Wykorzystano tu klasę `ReduceDriver`.

```

@Test
public void returnsMaximumIntegerInValues() throws IOException,
    InterruptedException {
    new ReduceDriver<Text, IntWritable, Text, IntWritable>()
        .withReducer(new MaxTemperatureReducer())
        .withInput(new Text("1950"),
            Arrays.asList(new IntWritable(10), new IntWritable(5)))
        .withOutput(new Text("1950"), new IntWritable(10))
        .runTest();
}

```

Kod tworzy listę wartości typu `IntWritable`, a następnie sprawdza, czy klasa `MaxTemperatureReducer` wybrała największą liczbę. Kod z listingu 6.9 to implementacja klasy `MaxTemperatureReducer` przechodzącej test.

*Listing 6.9. Reduktor z przykładu dotyczącego wyszukiwania maksymalnej temperatury*

```

public class MaxTemperatureReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {

        int maxValue = Integer.MIN_VALUE;
        for (IntWritable value : values) {
            maxValue = Math.max(maxValue, value.get());
        }
        context.write(key, new IntWritable(maxValue));
    }
}

```

## Uruchamianie kodu lokalnie na danych testowych

Skoro mapper i reduktor działają już dla kontrolowanych danych wejściowych, następny krok polega na napisaniu sterownika zadania i uruchomieniu go na danych testowych na maszynie programisty.

## Uruchamianie zadania w lokalnym mechanizmie wykonywania zadań

Za pomocą przedstawionego wcześniej w rozdziale interfejsu `Tool` łatwo jest napisać sterownik do wykonywania w modelu MapReduce zadania wyszukującego maksymalną temperaturę z poszczególnych lat (zobacz klasę `MaxTemperatureDriver` na listingu 6.10).

Listing 6.10. Aplikacja do wyszukiwania maksymalnej temperatury

```
public class MaxTemperatureDriver extends Configured implements Tool {

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.printf("Użytkowanie: %s [ogólne opcje] <wejście> <wyjście>\n",
                getClass().getSimpleName());
            ToolRunner.printGenericCommandUsage(System.err);
            return -1;
        }

        Job job = new Job(getConf(), "Max temperature");
        job.setJarByClass(getClass());

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapperClass(MaxTemperatureMapper.class);
        job.setCombinerClass(MaxTemperatureReducer.class);
        job.setReducerClass(MaxTemperatureReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new MaxTemperatureDriver(), args);
        System.exit(exitCode);
    }
}
```

Klasa `MaxTemperatureDriver` implementuje interfejs `Tool`, dzięki czemu można ustawiać opcje obsługiwane przez klasę `GenericOptionsParser`. Metoda `run()` tworzy obiekt typu `Job` na podstawie konfiguracji jednostki używanej do uruchamiania zadania. Spośród dostępnych parametrów konfiguracji zadania ustawiane są ścieżki do plików wejściowego i wyjściowego, klasy mappera, reduktora i mechanizmu łączenia (ang. *combiner*), a także typy wyjściowe (typy wejściowe są ustalane na podstawie formatu wejściowego; domyślnie używany jest format `TextInputFormat` z kluczami typu `LongWritable` i wartościami typu `Text`). Warto też ustawić nazwę zadania (`Max temperature`), co pozwala znaleźć zadanie na liście w trakcie jego wykonywania i po zakończeniu jego pracy. Domyślnie nazwą jest nazwa pliku JAR, która zwykle nie opisuje dobrze zadania.

Teraz można uruchomić aplikację na plikach lokalnych. Hadoop udostępnia lokalny mechanizm wykonywania zadań, który jest uproszczoną wersją silnika wykonawczego modelu MapReduce, przeznaczoną do uruchamiania zadań w jednej maszynie JVM. Ten mechanizm jest zaprojektowany na potrzeby testów i bardzo łatwo używa się go w środowisku IDE, ponieważ można uruchomić go w debugerze, aby w trybie kroczenia przejść przez kod mappera i reduktora.

Lokalny mechanizm wykonywania zadań jest używany, gdy właściwość `mapreduce.framework.name` jest ustawiona na wartość `local` (jest to ustawienie domyślne)<sup>1</sup>.

Z poziomu wiersza poleceń można uruchomić sterownik w następujący sposób:

```
% mvn compile
% export HADOOP_CLASSPATH=target/classes/
% hadoop v2.MaxTemperatureDriver -conf conf/hadoop-local.xml \
  input/ncdc/micro output
```

Inna możliwość to zastosowanie opcji `-fs` i `-jt` udostępnianych przez klasę `GenericOptionsParser`.

```
% hadoop v2.MaxTemperatureDriver -fs file:/// -jt local input/ncdc/micro output
```

Ta instrukcja uruchamia sterownik `MaxTemperatureDriver` z wykorzystaniem danych wejściowych z lokalnego katalogu `input/ncdc/micro` i generuje dane wyjściowe w lokalnym katalogu `output`. Zauważ, że choć ustawiona jest opcja `-fs` i używany jest lokalny system plików (`file:///`), lokalny mechanizm wykonywania zadań działa poprawnie dla dowolnego systemu plików, w tym dla systemu HDFS (jest to wygodne, jeśli przechowujesz w nim sporo plików).

Następnie można sprawdzić dane wyjściowe w lokalnym systemie plików:

```
% cat output/part-r-00000
1949 111
1950 22
```

## Testowanie sterownika

Zaimplementowanie w aplikacji interfejsu `Tool` zwiększa możliwości w zakresie opcji konfiguracyjnych, a ponadto ułatwia testowanie, ponieważ pozwala wstrzyknąć dowolny obiekt typu `Configuration`. Możesz to wykorzystać do napisania testu, który używa lokalnego mechanizmu wykonywania zadania na znanych danych wejściowych i sprawdza, czy dane wyjściowe są zgodne z oczekiwaniami.

Rozwiązanie można przygotować na dwa sposoby. Pierwszy polega na użyciu lokalnego mechanizmu do uruchomienia zadania na pliku testowym w lokalnym systemie plików. Kod z listingu 6.11 pokazuje, jak to zrobić.

*Listing 6.11. Test sterownika `MaxTemperatureDriver` z wykorzystaniem lokalnego, wewnątrzprocesowego mechanizmu wykonywania zadań*

```
@Test
public void test() throws Exception {
    Configuration conf = new Configuration();
    conf.set("fs.defaultFS", "file:///");
    conf.set("mapreduce.framework.name", "local");
    conf.setInt("mapreduce.task.io.sort.mb", 1);

    Path input = new Path("input/ncdc/micro");
    Path output = new Path("output");

    FileSystem fs = FileSystem.getLocal(conf);
    fs.delete(output, true); // Usuwanie dawnych danych wyjściowych

    MaxTemperatureDriver driver = new MaxTemperatureDriver();
```

<sup>1</sup> W Hadoopie 1 sposób wykonywania określa właściwość `mapred.job.tracker`. Wartość `local` oznacza użycie lokalnego mechanizmu wykonywania zadań, a rozdzielona przecinkiem para `host-port` pozwala podać adres jobtrackera.

```

driver.setConf(conf);

int exitCode = driver.run(new String[] {
    input.toString(), output.toString() });
assertThat(exitCode, is(0));

checkOutput(conf, output);
}

```

Test bezpośrednio ustawia opcje `fs.defaultFS` i `mapreduce.framework.name`, co oznacza, że używany jest lokalny system plików i mechanizm wykonywania zadań. Następnie test uruchamia sterownik `MaxTemperatureDriver` za pomocą interfejsu `Tool` na małym zbiorze znanych danych. W końcowej części testu wywoływana jest metoda `checkOutput()` w celu porównania wiersz po wierszu rzeczywistych danych wyjściowych z oczekiwanymi.

Drugi sposób testowania sterownika polega na uruchomieniu go za pomocą miniklastra. Hadoop udostępnia zestaw klas testowych `MiniDFSCluster`, `MiniMRCluster` i `MiniYARNCluster`, które umożliwiają programowe tworzenie klastrów wewnątrzprocesowych. W odróżnieniu od lokalnego mechanizmu wykonywania zadań te klasy pozwalają na przeprowadzanie testów z użyciem kompletnych systemów HDFS, MapReduce i YARN. Pamiętaj też, że menedżery węzłów w miniklastrach uruchamiają odrębne maszyny JVM do wykonywania zadań, co utrudnia debugowanie.



Miniklastrer możesz też uruchomić z poziomu wiersza poleceń za pomocą następującej instrukcji:

```

% hadoop jar \
  $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-*-tests.jar \
  minicluster

```

Miniklastry są często używane w zautomatyzowanych pakietach testów Hadoopa, można je jednak stosować także do testowania kodu użytkowników. Klasa abstrakcyjna `ClusterMapReduceTestCase` Hadoopa to użyteczny punkt wyjścia do pisania takich testów. Ta klasa obsługuje uruchamianie (metoda `setUp()`) i zatrzymywanie (metoda `tearDown()`) wewnątrzprocesowych klastrów z systemami HDFS i YARN oraz generuje odpowiedni obiekt typu `Configuration` dostosowany do współpracy z takimi klastrami. W podklasach trzeba tylko umieścić dane w systemie HDFS (na przykład w wyniku skopiowania ich z lokalnego pliku), uruchomić zadanie w modelu MapReduce i sprawdzić, czy dane wyjściowe są zgodne z oczekiwanymi. Przykładowy kod znajdziesz w klasie `MaxTemperatureDriverMiniTest` dostępnej w plikach dołączonych do książki.

Tego rodzaju testy są używane w testach regresyjnych i stanowią przydatne repozytorium przypadków brzegowych w danych wejściowych razem z oczekiwanymi dla nich wynikami. Gdy natrafisz na kolejne przypadki testowe, wystarczy, że dodasz je do pliku wejściowego i odpowiednio zaktualizujesz plik z oczekiwanymi danymi wyjściowymi.

## Uruchamianie programów w klastrze

Gdy już jesteś zadowolony z działania programu dla małego testowego zbioru danych, możesz wypróbować kod na pełnym zbiorze danych w klastrze Hadoopa. Opis budowania w pełni rozproszonego klastra znajdziesz w rozdziale 10. Możesz też wykorzystać informacje z tego punktu i utworzyć pseudorozproszony klastrer.

## Tworzenie pakietu z zadaniem

Lokalny mechanizm wykonywania zadań używa jednej maszyny JVM do wykonywania zadania. Dlatego jeśli wszystkie klasy potrzebne w zadaniu są dostępne w ścieżce do klas, rozwiązanie będzie działać.

W środowisku rozproszonym sytuacja się komplikuje. Klasy zadania trzeba umieścić w *pliku JAR zadania* umieszczanym w klastrze. Hadoop automatycznie znajduje ten plik, wyszukując pliki JAR na podstawie ścieżki do klas sterownika zawierającej klasę ustawioną w metodzie `setJarByClass()` (klasy `JobConf` lub `Job`). Inna możliwość to ustawienie pliku JAR bezpośrednio za pomocą ścieżki do pliku, do czego służy metoda `setJar()`. Ścieżka do pliku JAR może prowadzić do lokalnego systemu plików lub systemu HDFS.

Do tworzenia pliku JAR można wykorzystać wygodne narzędzie do budowania, takie jak Ant lub Maven. Na podstawie pliku POM przedstawionego na listingu 6.3 poniższa instrukcja utworzy w katalogu projektu plik JAR *hadoop-examples.jar* zawierający wszystkie skompilowane klasy.

```
% mvn package -DskipTests
```

Jeśli każdemu zadaniu odpowiada plik JAR, można wskazać przeznaczoną do uruchomienia główną klasę w manifeście pliku JAR. Jeżeli główna klasa nie jest wskazana w manifeście, trzeba ją ustawić w wierszu poleceń (wkrótce w kontekście uruchamiania zadania zobaczysz, jak to zrobić).

Pliki JAR z zależnościami można umieścić w podkatalogu *lib* pliku JAR zadania, przy czym istnieją też inne, opisane dalej sposoby dodawania zależności. Podobnie pliki zasobów można zapisać w podkatalogu *classes* (to rozwiązanie przypomina pliki WAR — ang. *Web application archive* — Javy, przy czym pliki JAR trafiają w nich do podkatalogu *WEB-INF/lib*, a klasy do podkatalogu *WEB-INF/classes*).

### Ścieżka do klas po stronie klienta

Ścieżka do klas po stronie klienta ustawiana przez polecenie `hadoop jar <jar>` obejmuje następujące elementy:

- plik JAR zadania,
- pliki JAR z katalogu *lib* z pliku JAR zadania oraz katalog *classes* (jeśli istnieje),
- zawartość zmiennej `HADOOP_CLASSPATH` (jeżeli jest ustawiona).

To wyjaśnia, dlaczego w zmiennej `HADOOP_CLASSPATH` trzeba wskazać klasy i biblioteki z zależnościami, jeśli używany jest lokalny mechanizm uruchamiania zadań bez pliku JAR zadania (wywołanie `hadoop NAZWA_KLASY`).

### Ścieżka do klas dla operacji

W klastrze (dotyczy to także trybu pseudorozproszonego) operacje mapowania i redukcji działają w odrębnych maszynach JVM. Ścieżki do klas tych operacji *nie* są kontrolowane za pomocą zmiennej `HADOOP_CLASSPATH`. `HADOOP_CLASSPATH` to ustawienie używane po stronie klienta, określające tylko ścieżkę do klas maszyny JVM sterownika, która zgłasza zadanie.



Ścieżka do klas operacji użytkownika obejmuje następujące elementy:

- plik JAR zadania,
- pliki JAR z katalogu *lib* z pliku JAR zadania oraz katalog *classes* (jeśli istnieje),
- pliki dodane do rozproszonej pamięci podręcznej za pomocą opcji `-libjars` (zobacz tabelę 6.1) lub metody `addFileToClassPath()` klasy `DistributedCache` (w dawnym interfejsie API) albo `Job` (w nowym interfejsie API).

## Dodawanie zależności do pakietu

Ponieważ istnieją różne sposoby kontrolowania zawartości ścieżki do klas klientów i operacji, dostępne są też odpowiednie techniki dołączania zależności bibliotecznych dla zadań. Możesz:

- wypakować biblioteki i dodać je do pliku JAR zadania,
- umieścić biblioteki w katalogu *lib* pliku JAR zadania,
- przechowywać biblioteki niezależnie od pliku JAR zadania i dodawać je do ścieżki do klas klienta za pomocą zmiennej `HADOOP_CLASSPATH` i do ścieżki do klas operacji przy użyciu opcji `-libjars`.

Ostatnie rozwiązanie, wykorzystanie rozproszonej pamięci podręcznej, jest najprostsze, jeśli chodzi o budowanie, ponieważ zależności nie trzeba dodawać do pliku JAR zadania. Ponadto użycie rozproszonej pamięci podręcznej może oznaczać mniej operacji przenoszenia plików JAR w klastrze, ponieważ pliki między operacjami można przechowywać w pamięci podręcznej w węźle.

## Kolejność uwzględniania ścieżek do klas operacji

Pliki JAR użytkownika są dodawane na koniec ścieżek do klas klienta i operacji. W niektórych sytuacjach powoduje to konflikt zależności z wbudowanymi bibliotekami Hadoopa, jeśli Hadoop korzysta z innej, niezgodnej wersji biblioteki używanej w kodzie. Czasem potrzebna jest możliwość kontrolowania kolejności elementów w ścieżce do klas operacji, tak aby potrzebne klasy były wybierane jako pierwsze. Po stronie klienta można wymusić na Hadoopie umieszczenie na początku ścieżki do klas użytkownika. Aby to zrobić, ustaw zmienną środowiskową `HADOOP_USER_CLASSPATH_FIRST` na wartość `true`. Natomiast by na początku wybierana była ścieżka do klas operacji, na wartość `true` należy ustawić właściwość `mapreduce.job.user.classpath.first`. Zauważ, że ustawienie tych opcji powoduje zmianę kolejności wczytywanych klas z zależnościami platformy Hadoop (ale tylko w danym zadaniu). Potencjalnie może to spowodować błąd przesyłania zadania lub wykonywania operacji, dlatego zachowaj ostrożność przy stosowaniu tych opcji.

## Uruchamianie zadania

Aby wywołać zadanie, należy uruchomić sterownik, wskazując przy tym za pomocą opcji `-conf` klastrer, w którym zadanie ma działać (można też zastosować opcje `-fs` i `-jt`).

```
% unset HADOOP_CLASSPATH
% hadoop jar hadoop-examples.jar v2.MaxTemperatureDriver \
  -conf conf/hadoop-cluster.xml input/ncdc/all max-temp
```



Zmienna środowiskowa `HADOOP_CLASSPATH` jest zerowana, ponieważ dla tego zadania nie są używane żadne niestandardowe zależności. Gdyby zmienna pozostała ustawiona na wartość `target/classes/` (tę wartość dodano wcześniej w rozdziale), Hadoop nie znalazłby pliku JAR z zadaniem. Wczytałby klasę `MaxTemperatureDriver` z katalogu `target/classes` zamiast z pliku JAR, przez co zadanie zakończyłoby się niepowodzeniem.

Metoda `waitForCompletion()` klasy `Job` uruchamia zadanie i sprawdza jego postęp. Gdy w operacji mapowania lub redukowania zachodzą zmiany, wspomniana metoda wyświetla informacje na ten temat. Oto dane wyjściowe (część wierszy w celu zachowania przejrzystości usunięto).

```
14/09/12 06:38:11 INFO input.FileInputFormat: Total input paths to process : 101
14/09/12 06:38:11 INFO impl.YarnClientImpl: Submitted application
application_1410450250506_0003
14/09/12 06:38:12 INFO mapreduce.Job: Running job: job_1410450250506_0003
14/09/12 06:38:26 INFO mapreduce.Job: map 0% reduce 0%
...
14/09/12 06:45:24 INFO mapreduce.Job: map 100% reduce 100%
14/09/12 06:45:24 INFO mapreduce.Job: Job job_1410450250506_0003 completed
successfully
14/09/12 06:45:24 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=93995
    FILE: Number of bytes written=10273563
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=33485855415
    HDFS: Number of bytes written=904
    HDFS: Number of read operations=327
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=16
  Job Counters
    Launched map tasks=101
    Launched reduce tasks=8
    Data-local map tasks=101
    Total time spent by all maps in occupied slots (ms)=5954495
    Total time spent by all reduces in occupied slots (ms)=74934
    Total time spent by all map tasks (ms)=5954495
    Total time spent by all reduce tasks (ms)=74934
    Total vcore-seconds taken by all map tasks=5954495
    Total vcore-seconds taken by all reduce tasks=74934
    Total megabyte-seconds taken by all map tasks=6097402880
    Total megabyte-seconds taken by all reduce tasks=76732416
  Map-Reduce Framework
    Map input records=1209901509
    Map output records=1143764653
    Map output bytes=10293881877
    Map output materialized bytes=14193
    Input split bytes=14140
    Combine input records=1143764772
    Combine output records=234
    Reduce input groups=100
    Reduce shuffle bytes=14193
    Reduce input records=115
    Reduce output records=100
    Spilled Records=379
    Shuffled Maps =808
    Failed Shuffles=0
    Merged Map outputs=808
    GC time elapsed (ms)=101080
```

```
CPU time spent (ms)=5113180
Physical memory (bytes) snapshot=60509106176
Virtual memory (bytes) snapshot=167657209856
Total committed heap usage (bytes)=68220878848
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=33485841275
File Output Format Counters
  Bytes Written=90
```

Te dane wyjściowe obejmują wiele przydatnych informacji. Przed rozpoczęciem zadania wyświetlany jest jego identyfikator. Jest on potrzebny, gdy trzeba wskazać zadanie (na przykład w plikach dziennika) lub przy sprawdzaniu go przy użyciu polecenia `mapred job`. Po zakończeniu pracy zadania wyświetlane są dotyczące go statystyki (nazywane licznikami — *counters*). Są one bardzo przydatne do upewniania się, że zadanie wykonało oczekiwane czynności. Na przykład w pokazanym zadaniu przeanalizowano 1,2 miliarda rekordów (`Map input records`) i wczytano dane z około 34 GB skompresowanych plików z systemu HDFS (`HDFS: Number of bytes read`). Dane wejściowe zostały podzielone na 101 plików w formacie gzip o akceptowalnej wielkości, co oznacza, że nie wystąpiły problemy z ich podziałem.

Więcej o znaczeniu poszczególnych liczników dowiesz się z punktu „Liczniki wbudowane” w rozdziale 9.

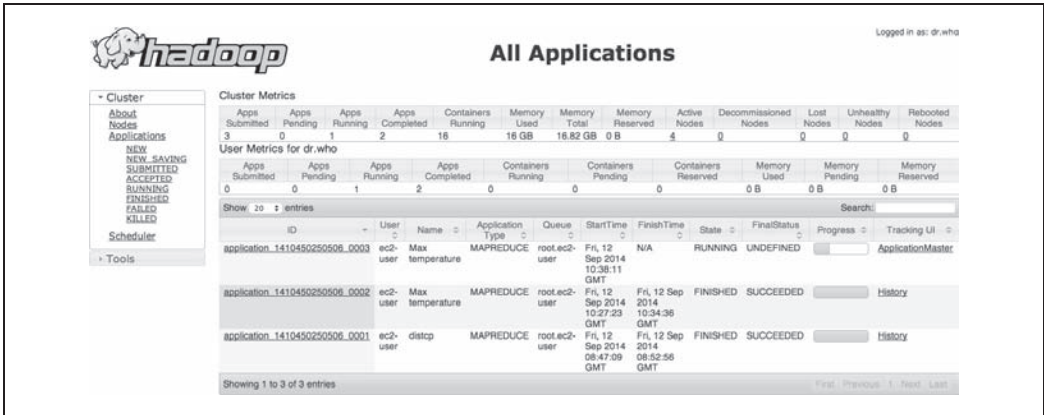
## Sieciowy interfejs użytkownika modelu MapReduce

Hadoop udostępnia sieciowy interfejs użytkownika do wyświetlania informacji na temat zadań. Jest on przydatny do śledzenia postępów zadań w trakcie ich wykonywania, a także wyszukiwania statystyk i dzienników po zakończeniu pracy przez zadanie. Ten interfejs użytkownika jest dostępny pod adresem <http://resource-manager-host:8088/>.

### Strona menedżera zasobów

Zrzut strony głównej jest przedstawiony na rysunku 6.1. Sekcja *Cluster Metrics* zawiera przegląd informacji o klastrze. Obejmują one liczbę aplikacji aktualnie działających w klastrze (oraz aplikacji w innym stanie), ilość dostępnych w klastrze zasobów (*Memory Total*) i informacje o menedżerach węzłów.

W głównej tabeli wymienione są wszystkie aplikacje, które zostały wykonane lub działają w klastrze. Dostępne jest pole wyszukiwania przydatne do filtrowania aplikacji w celu znalezienia potrzebnych. W głównym oknie wyświetlanych jest do 100 pozycji na stronę, a menedżer zasobów przechowuje w pamięci jednocześnie do 10 000 ukończonych aplikacji (tę wartość określa właściwość `yarn.resourcemanager.max-completed-applications`); starsze aplikacje są dostępne tylko na stronie historii zadań. Zauważ, że historia zadań jest trwała, dlatego znajdują się w niej także zadania z poprzednich uruchomień menedżera zasobów.



Rysunek 6.1. Zrzut strony menedżera zasobów

## Identyfikatory zadania, operacji i prób wykonania operacji

W Hadoopie 2 identyfikatory zadań w modelu MapReduce są generowane na podstawie identyfikatorów aplikacji systemu YARN (te identyfikatory są tworzone przez menedżer zasobów systemu YARN). Identyfikator aplikacji obejmuje czas uruchomienia menedżera zasobów (a nie samej aplikacji) i wartość licznika zwiększaną przez menedżer zasobów. Wartość licznika pozwala w niepowtarzalny sposób identyfikować aplikacje w danym egzemplarzu menedżera zasobów. Oto przykładowy identyfikator aplikacji:

```
application_1410450250506_0003
```

Jest to trzecia (0003; identyfikatory aplikacji rozpoczynają się od wartości 1) aplikacja z danego menedżera zasobów, który rozpoczął pracę w czasie reprezentowanym przez znacznik 1410450250506. Licznik jest uzupełniany początkowymi zerami, co pozwala na wygodne sortowanie identyfikatorów (na przykład na liście zawartości katalogów). Jednak gdy licznik dojdzie do wartości 10000, *nie* jest resetowany, przez co identyfikatory aplikacji stają się dłuższe. Utrudnia to sortowanie.

Identyfikator zadania jest tworzony w wyniku zastąpienia przedrostka `application` z identyfikatora aplikacji przedrostkiem `job`.

```
job_1410450250506_0003
```

Operacje należą do zadania, a ich identyfikatory są tworzone przez zastąpienie przedrostka `job` w identyfikatorze zadania przedrostkiem `task` i dodanie przyrostka identyfikującego operację w zadaniu. Oto przykład:

```
task_1410450250506_0003_m_000003
```

Jest to czwarta (000003; identyfikatory operacji są numerowane od 0) operacja mapowania (m) w zadaniu o identyfikatorze `job_1410450250506_0003`. Identyfikatory operacji są tworzone dla zadania w momencie ich inicjowania, dlatego nie wyznaczają kolejności wykonywania operacji.

Operacje mogą być wykonywane więcej niż raz, na przykład z powodu błędu (zobacz punkt „Niepowodzenie operacji” w rozdziale 7.) lub spekulacyjnego wykonania (zobacz punkt „Wykonywanie spekulacyjne” w rozdziale 7.). Dlatego aby zidentyfikować różne przypadki wykonania operacji, nadaje się unikatowe identyfikatory próbom wykonania operacji. Oto przykład:

```
attempt_1410450250506_0003_m_000003_0
```

Jest to pierwsza (0; identyfikatory prób wykonania są numerowane od 0) próba uruchomienia operacji `task_1410450250506_0003_m_000003`. Próby wykonania operacji są podejmowane w zależności od potrzeby w trakcie pracy zadania, dlatego ich numery odpowiadają kolejności podejmowania prób.

## Historia zadań

**Historia zadań** dotyczy zdarzeń i konfiguracji ukończonych zadań w modelu MapReduce. Jest zachowywana niezależnie od tego, czy zadanie zostało ukończone z powodzeniem, czy nie. Ma zapewniać przydatne informacje użytkownikowi uruchamiającemu zadanie.

Pliki z historią zadań są zapisywane przez zarządcę aplikacji z modelu MapReduce w systemie HDFS w katalogu określonym we właściwości `mapreduce.jobhistory.done-dir`. Pliki historii zadań są przechowywane przez tydzień, po czym są usuwane przez system.

Dziennik z historią obejmuje zdarzenia dotyczące zadań, operacji i prób. Wszystkie te informacje są przechowywane w pliku w formacie JSON. Historię konkretnego zadania można wyświetlić za pomocą sieciowego interfejsu użytkownika serwera historii zadań (odnośnik do niego znajdziesz na stronie menedżera zasobów) lub z poziomu wiersza poleceń przy użyciu instrukcji `mapred job -history` (należy jej wskazać plik z historią zadań).

### Strona zadania w modelu MapReduce

Gdy klikniesz odnośnik *Tracking UI*, przejdziesz do sieciowego interfejsu użytkownika zarządcy aplikacji (lub do strony z historią, jeśli aplikacja zakończyła pracę). W modelu MapReduce pojawia się wtedy strona zadania taka jak na rysunku 6.2.

MapReduce Job  
job\_1410450250506\_0003

Job Overview

Job Name: Max temperature  
State: RUNNING  
Uberized: false  
Started: Fri Sep 12 06:38:24 EDT 2014  
Elapsed: 6mins, 25sec

ApplicationMaster	Attempt Number	Start Time	Node	Logs
1		Fri Sep 12 06:38:19 EDT 2014	ip-10-1-1-172.ec2.internal:8042	logs

Task Type	Progress	Total	Pending	Running	Complete
Map	<div style="width: 25%;"></div>	101	25	14	62
Reduce	<div style="width: 8%;"></div>	8	8	0	0

Attempt Type	New	Running	Failed	Killed	Successful
Maps	25	14	0	0	62
Reduces	8	0	0	0	0

Rysunek 6.2. Zrzuty strony zadania

W trakcie działania zadania można na tej stronie śledzić jego postęp. Tabela w dolnej części strony pokazuje postęp operacji mapowania i redukowania. Kolumna *Total* zawiera łączną liczbę operacji mapowania i redukowania dla danego zadania (dla poszczególnych typów operacji dostępne są odrębne wiersze). Pozostałe kolumny wyświetlają stan operacji — *Pending* (oczekujące na uruchomienie), *Running* (w toku) lub *Complete* (z powodzeniem ukończone).

W dolnej części tabeli widoczna jest łączna liczba nieudanych i zamkniętych prób wykonania operacji mapowania i redukowania. Próby wykonania operacji są oznaczone jako zamknięte, jeśli są duplikatami spekulacyjnych prób wykonania, jeśli ich węzeł przestał działać lub jeśli zostały zamknięte przez użytkownika. Omówienie nieudanych wykonań operacji znajdziesz w punkcie „Niepowodzenie operacji” w rozdziale 7.

Ponadto w nawigacji znajduje się wiele przydatnych odnośników. Na przykład odnośnik *Configuration* prowadzi do skonsolidowanego pliku z konfiguracją zadania, zawierającego wszystkie właściwości i ich wartości z czasu wykonywania zadania. Jeśli nie masz pewności, na jaką wartość ustawiona była konkretna właściwość, możesz kliknąć ten odnośnik i sprawdzić wspomniany plik.

## Pobieranie wyników

Po zakończeniu wykonywania zadania wyniki można pobrać na kilka sposobów. Każdy reduktor generuje jeden plik wyjściowy. W katalogu *max-temp* pojawia się więc 30 fragmentarycznych plików o nazwach od *part-r-00000* do *part-r-00029*.



Jak wskazuje na to nazwa, fragmentaryczne pliki warto traktować jak fragmenty „pliku” *max-temp*.

Jeśli dane wyjściowe są długie (inaczej niż w omawianym przykładzie), ważne jest używanie wielu fragmentów, dzięki czemu równoległe mogą działać liczne reduktory. Nawet jeśli plik jest podzielony na części, zwykle nadal można z niego łatwo korzystać — na przykład jak z danych wejściowych do innego zadania w modelu MapReduce. W niektórych sytuacjach można wykorzystać strukturę fragmentów do przeprowadzenia złączenia na etapie mapowania (zobacz punkt „Złączanie po stronie mapowania” w rozdziale 9.).

Omawiane zadanie generuje bardzo niewielką ilość danych wyjściowych, dlatego można je wygodnie skopiować z systemu HDFS do maszyny programisty. Służy do tego opcja `-getmerge` instrukcji `hadoop fs`, która pobiera wszystkie pliki z katalogu wskazanego za pomocą wzorca wejściowego i scala je w jeden plik w lokalnym systemie plików.

```
% hadoop fs -getmerge max-temp max-temp-local
% sort max-temp-local | tail
1991      607
1992      605
1993      567
1994      568
1995      567
1996      561
1997      565
1998      568
1999      568
2000      558
```

Dane wyjściowe są sortowane, ponieważ fragmenty z danych wyjściowych z etapu redukcji są nieuporządkowane (to efekt działania partycjonującej funkcji skrótu). Przetwarzanie końcowe danych z modelu MapReduce jest bardzo częste, podobnie jak przekazywanie ich do narzędzi analitycznych (takich jak R, arkusze kalkulacyjne, a nawet relacyjne bazy danych).

Inny sposób na pobieranie niewielkich zbiorów danych wyjściowych to użycie opcji `-cat` w celu wyświetlenia plików wyjściowych w konsoli.

```
% hadoop fs -cat max-temp/*
```

Po bliższej analizie okazuje się, że niektóre wyniki wyglądają podejrzanie. Na przykład maksymalna temperatura dla roku 1951 (niepokazana w wynikach) to 590°C! Jak ustalić, co jest tego przyczyną? Czy to efekt błędnych danych wejściowych, czy błędu w programie?

## Debugowanie zadania

Tradycyjnym sposobem debugowania programów jest używanie instrukcji `print`. Oczywiście w Hadoopie też jest to możliwe. Należy jednak uwzględnić różne komplikacje. Programy działają w dziesiątkach, setkach, a nawet tysiącach węzłów. Jak więc znaleźć i zbadać dane wyjściowe z instrukcji diagnostycznych z rozmaitych węzłów? W omawianym przykładzie szukany jest nietypowy (zdaniem programisty) przypadek. Dlatego można wykorzystać instrukcję diagnostyczną do zarejestrowania informacji w standardowym strumieniu błędów i zaktualizować komunikat o stanie operacji, tak by sugerował zapoznanie się z dziennikiem błędów. Sieciowy interfejs użytkownika pozwala łatwo uzyskać pożądaną efekt dzięki przekazaniu odpowiedniego tekstu.

Można też utworzyć niestandardowy licznik określający łączną liczbę rekordów z nierealistycznymi temperaturami w całym zbiorze danych. Zapewnia to cenne informacje na temat tego, jak radzić sobie z takimi danymi. Jeśli się okaże, że błędne temperatury pojawiają się często, konieczne może okazać się sprawdzenie, z czego to wynika, i wymyślenie sposobu na pobieranie temperatury zamiast prostego ignorowania podejrzanych rekordów. W trakcie diagnozowania zadania zawsze należy zadać sobie pytanie, czy można wykorzystać licznik do uzyskania informacji potrzebnych do ustalenia, co się dzieje. Nawet jeśli konieczne są komunikaty o stanie lub rejestrowanie zdarzeń, licznik może okazać się przydatny do oceny powagi problemu. Więcej o licznikach dowiesz się z punktu „Liczniki” w rozdziale 9.

Jeśli w trakcie diagnozowania generowane są duże ilości danych w dzienniku, dostępnych jest kilka możliwości. Jedna polega na zapisaniu informacji w danych wyjściowych etapu mapowania zamiast w standardowym strumieniu błędów. Pozwala to na analizę i agregację danych w ramach operacji redukcji. To podejście zwykle wymaga zmian w strukturze programu, dlatego warto zacząć od innej techniki. Polega ona na napisaniu programu (oczywiście w modelu MapReduce) do analizowania generowanych przez zadanie danych z dziennika.

Kod związany z diagnozowaniem należy dodać do mappera (wersja 3.), a nie do reduktora, ponieważ celem jest ustalenie danych źródłowych prowadzących do uzyskania nieakceptowalnych danych wyjściowych.

```
public class MaxTemperatureMapper
    extends Mapper<LongWritable, Text, Text, IntWritable> {

    enum Temperature {
        OVER_100
    }

    private NcdcRecordParser parser = new NcdcRecordParser();

    @Override
    public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {

        parser.parse(value);
        if (parser.isValidTemperature()) {
            int airTemperature = parser.getAirTemperature();
            if (airTemperature > 1000) {
                System.err.println("Temperatura ponad 100 stopni dla danych: " + value);
                context.setStatus("Wykryto uszkodzony rekord (patrz dziennik).");
                context.getCounter(Temperature.OVER_100).increment(1);
            }
        }
    }
}
```



```

        context.write(new Text(parser.getYear()), new IntWritable(airTemperature));
    }
}
}

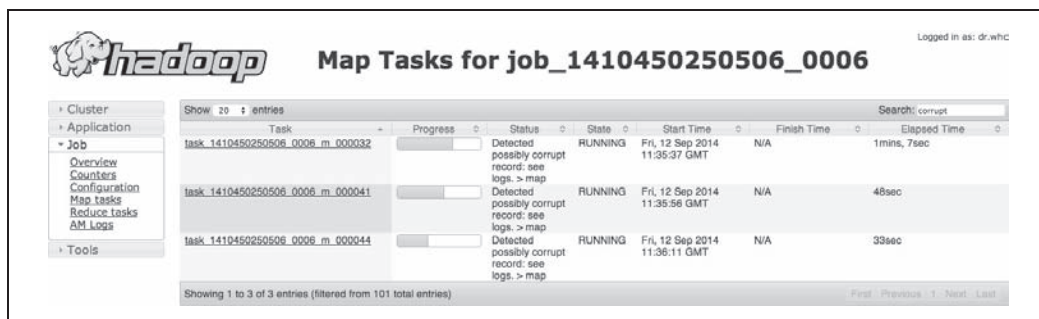
```

Jeśli temperatura wynosi ponad 100°C (co reprezentują wartości powyżej 1000, ponieważ temperatura jest zapisywana w dziesiętnych częściach stopni), kod zapisuje w standardowym strumieniu błędów podejrzany wiersz, a także za pomocą metody setStatus() klasy Context aktualizuje komunikat o stanie operacji mapowania, by skierować użytkowników do dziennika. Ponadto zwiększona jest wartość licznika, reprezentowana w Javie za pomocą pola typu wyliczeniowego. W tym programie zdefiniowane jest jedno pole, OVER\_100, pozwalające zliczyć rekordy z temperaturą przekraczającą 100°C.

Po wprowadzeniu zmian należy ponownie skompilować kod, odtworzyć plik JAR, a następnie uruchomić zadanie i w trakcie jego wykonywania przejść do strony operacji.

### Strony operacji i prób wykonania operacji

Strona zadania zawiera szereg odnośników do wyświetlania szczegółowych danych na temat operacji. Na przykład odnośnik *Map* prowadzi do strony z informacjami o wszystkich operacjach mapowania. Zrzut z rysunku 6.3 przedstawia tę stronę dla uruchomionego zadania. W kolumnie *Status* operacji widoczne są komunikaty diagnostyczne.



Rysunek 6.3. Zrzut ze strony operacji

Kliknięcie odnośnika do operacji powoduje przejście do strony z próbami wykonania. Widoczne są na niej wszystkie próby wykonania danej operacji. Przy każdej próbie znajdują się odnośniki do plików dziennika i liczników. Kliknięcie jednego z odnośników do plików dziennika dla udanych prób wykonania zadania pozwala znaleźć zarejestrowane podejrzane rekordy z danych wyjściowych (pokazany wiersz jest podzielony i skrócony, aby zmieścił się na stronie).

```

Temperatura ponad 100 stopni dla danych:
0335999999433181957042302005+37950+139117SAO +0004RJSN V02011359003150070356999
999433201957010100005+35317+139650SAO +000899999V02002359002650076249N0040005...

```

Wygląda na to, że ten rekord ma inny format niż pozostałe. W wierszu występują spacje, co nie jest opisane w specyfikacji.

Po zakończeniu zadania można sprawdzić wartość zdefiniowanego licznika, aby ustalić, ile rekordów z temperaturą powyżej 100°C znajduje się w całym zbiorze danych. Dostęp do liczników jest możliwy za pomocą sieciowego interfejsu użytkownika i wiersza poleceń.



```
% mapred job -counter job_1410450250506_0006 \  
'v3.MaxTemperatureMapper$Temperature' OVER_100
```

3

Opcja `-counter` przyjmuje identyfikator zadania, nazwę grupy licznika (tu jest to pełna nazwa klasy) i nazwę licznika (nazwę wyliczenia). W całym zbiorze danych obejmującym ponad miliard rekordów występują tylko trzy błędne rekordy. Usuwanie nieprawidłowych rekordów to standardowe rozwiązanie w wielu problemach związanych z wielkimi danymi, choć tu należy zachować ostrożność, ponieważ szukane są skrajne wartości (maksymalna temperatura), a nie zagregowany wynik. Jednak pominięcie trzech rekordów prawdopodobnie nie zmieni rezultatów.

## Obsługa nieprawidłowych danych

Zapisanie danych wejściowych powodujących problem jest przydatne. Można je wykorzystać w teście do sprawdzenia, czy mapper działa prawidłowo. Poniższy test z platformy MRUnit sprawdza, czy licznik jest aktualizowany po wykryciu błędnych danych wejściowych.

```
@Test  
public void parsesMalformedTemperature() throws IOException,  
    InterruptedException {  
    Text value = new Text("0335999999433181957042302005+37950+139117SAO +0004" +  
        // Rok ~~~~  
        "RJSN V02011359003150070356999999433201957010100005+353");  
        // Temperatura ~~~~~  
    Counters counters = new Counters();  
    new MapDriver<LongWritable, Text, Text, IntWritable>()  
        .withMapper(new MaxTemperatureMapper())  
        .withInput(new LongWritable(0), value)  
        .withCounters(counters)  
        .runTest();  
    Counter c = counters.findCounter(MaxTemperatureMapper.Temperature.MALFORMED);  
    assertThat(c.getValue(), is(1L));  
}
```

Rekord powodujący problem ma odmienny format niż inne przedstawione wcześniej wiersze. Na listingu 6.12 pokazany jest zmodyfikowany program (wersja 4.). Używa on parsera ignorującego każdy wiersz, w którym pole z temperaturą nie jest poprzedzone znakiem plus lub minus. Ponadto dodany został licznik, który określa, ile rekordów zostało pominiętych z tego powodu.

Listing 6.12. Mapper dla przykładu z wyszukiwaniem maksymalnej temperatury

```
public class MaxTemperatureMapper  
    extends Mapper<LongWritable, Text, Text, IntWritable> {  
  
    enum Temperature {  
        MALFORMED  
    }  
  
    private NcdcRecordParser parser = new NcdcRecordParser();  
  
    @Override  
    public void map(LongWritable key, Text value, Context context)  
        throws IOException, InterruptedException {  
  
        parser.parse(value);  
        if (parser.isValidTemperature()) {  
            int airTemperature = parser.getAirTemperature();  
            context.write(new Text(parser.getYear()), new IntWritable(airTemperature));  
        }  
    }  
}
```

```

    } else if (parser.isMalformedTemperature()) {
        System.err.println("Ignorowanie potencjalnie błędnych danych: " + value);
        context.getCounter(Temperature.MALFORMED).increment(1);
    }
}
}
}

```

## Dzienniki w Hadoopie

Hadoop generuje dzienniki w różnych miejscach i dla rozmaitych odbiorców. Przegląd dzienników przedstawia tabela 6.2.

Tabela 6.2. Typy dzienników w Hadoopie

Dzienniki	Główni odbiorcy	Opis	Dodatkowe informacje
Dzienniki demonów systemowych	Administratorzy	Każdy demon w Hadoopie generuje plik dziennika (za pomocą narzędzia log4j) i plik łączący standardowe wyjście i wyjście błędów. Są one zapisywane w katalogu zdefiniowanym przez zmienną środowiskową HADOOP_LOG_DIR.	„Pliki dzienników systemowych” w rozdziale 10. i „Rejestrowanie informacji w dziennikach” w rozdziale 11.
Dzienniki inspekcji z systemu HDFS	Administratorzy	Dziennik wszystkich żądań z systemu HDFS (domyślnie wyłączony). Zapisywany w dzienniku węzła nazw, choć można to zmienić.	„Rejestrowanie dziennika inspekcji” w rozdziale 11.
Dzienniki historii zadań z modelu MapReduce	Użytkownicy	Dziennik zdarzeń (takich jak ukończenie operacji) zachodzących w czasie wykonywania zadania. Zapisywany centralnie w systemie HDFS.	„Historia zadań”
Dzienniki operacji z modelu MapReduce	Użytkownicy	Każdy proces potomny operacji generuje za pomocą narzędzia log4j plik dziennika ( <i>syslog</i> ), plik danych przesyłanych do standardowego wyjścia ( <i>stdout</i> ) i plik standardowego strumienia błędów ( <i>stderr</i> ). Są one zapisywane w podkatalogu <i>userlogs</i> katalogu zdefiniowanego w zmiennej środowiskowej YARN_LOG_DIR.	Ten punkt

System YARN udostępnia usługę **agregowania dzienników**, która przyjmuje dzienniki operacji z ukończonych aplikacji i przynosi je do systemu HDFS, gdzie są zapisywane w archiwalnym pliku kontenerowym. Jeśli usługa jest włączona (wymaga to ustawienia w klastrze właściwości `yarn.log-aggregation-enable` na wartość `true`), dzienniki operacji można wyświetlić za pomocą odnośnika `logs` z sieciowego interfejsu użytkownika dla prób wykonania operacji lub przy użyciu polecenia `mapred job -logs`.

Agregowanie dzienników domyślnie nie jest włączone. Wtedy dzienniki operacji można pobrać za pomocą sieciowego interfejsu użytkownika menedżera węzła (`http://node-manager-host:8042/logs/userlogs`).

Zapis danych w plikach dzienników jest prosty. Wszystko, co jest zapisywane do standardowego wyjścia lub standardowego strumienia błędów, trafia do odpowiedniego pliku dziennika. Oczywiście w narzędziu Streaming standardowe wyjście jest używane dla danych wyjściowych operacji mapowania lub redukcji, dlatego te dane nie pojawiają się w dzienniku standardowego wyjścia.

W Javie możesz zapisać informacje w pliku *syslog* operacji za pomocą interfejsu API Apache Commons Logging (lub dowolnego interfejsu API do rejestrowania dzienników współdziałającego z narzędziem log4j). To podejście zastosowano na listingu 6.13.

Listing 6.13. Mapper tożsamościowy zapisujący dane do standardowego wyjścia i używający interfejsu API Apache Commons Logging

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.hadoop.mapreduce.Mapper;

public class LoggingIdentityMapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT>
    extends Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {

    private static final Log LOG = LogFactory.getLog(LoggingIdentityMapper.class);

    @Override
    @SuppressWarnings("unchecked")
    public void map(KEYIN key, VALUEIN value, Context context)
        throws IOException, InterruptedException {
        // Rejestrowanie w pliku stdout
        System.out.println("Klucz z etapu mapowania: " + key);

        // Rejestrowanie w pliku syslog
        LOG.info("Klucz z etapu mapowania: " + key);
        if (LOG.isDebugEnabled()) {
            LOG.debug("Wartość z etapu mapowania: " + value);
        }
        context.write((KEYOUT) key, (VALUEOUT) value);
    }
}
```

Domyślny poziom rejestrowania zdarzeń to INFO. Dlatego komunikaty z poziomu DEBUG nie pojawiają się w pliku dziennika *syslog* operacji. Jednak czasem programista chce wyświetlać takie komunikaty. W tym celu należy ustawić właściwość `mapreduce.map.log.level` lub `mapreduce.reduce.log.level`. Poniższa instrukcja pokazuje, jak ustawić właściwość dla mappera, aby zobaczyć w dzienniku wartości z etapu mapowania.

```
% hadoop jar hadoop-examples.jar LoggingDriver -conf conf/hadoop-cluster.xml \  
-D mapreduce.map.log.level=DEBUG input/ncdc/sample.txt logging-out
```

Istnieją narzędzia do zmiany czasu przechowywania i wielkości dzienników operacji. Domyślnie dzienniki są usuwane po upływie przynajmniej trzech godzin (tę wartość można ustawić za pomocą właściwości `yarn.nodemanager.log.retain-seconds`, choć jest ona ignorowana, jeśli włączone jest agregowanie dzienników). Można też ustawić limit maksymalnej wielkości każdego dziennika. Służy do tego właściwość `mapreduce.task.userlog.limit.kb`. Jej wartość domyślna, 0, oznacza brak limitu.



Czasem programista musi zdiagnozować problem, który prawdopodobnie występuje w maszynie JVM uruchamiającej polecenie Hadoopa, a nie w klastrze. Aby przesłać dzienniki z poziomu DEBUG do konsoli, należy użyć instrukcji takiej jak poniżej.

```
% HADOOP_ROOT_LOGGER=DEBUG,console hadoop fs -text /foo/bar
```

## Zdalne diagnozowanie

Gdy operacja kończy się niepowodzeniem i zarejestrowanych jest za mało informacji, aby można było zdiagnozować problem, można uruchomić debugger dla danej operacji. Trudno jest to zrobić, jeśli zadanie działa w klastrze. Nie wiadomo wtedy, które węzły będą przetwarzać poszczególne fragmenty danych wejściowych, dlatego nie można przygotować debugera przed wystąpieniem awarii. Istnieje jednak kilka rozwiązań.

### *Odtworzenie awarii lokalnie*

W operacjach problemy często występują dla określonych danych wejściowych. Można spróbować odtworzyć problem lokalnie, pobierając plik, dla którego operacja kończy się niepowodzeniem, i uruchamiając zadanie lokalnie (na przykład za pomocą debugera VisualVM Javy).

### *Użycie opcji diagnostycznych maszyny JVM*

Częstym powodem awarii jest błąd braku pamięci z Javy występujący w maszynie JVM operacji. Możesz dodać do elementu `mapred.child.java.opts` ustawienie `-XX:-HeapDumpOnOutOfMemoryError-XX:HeapDumpPath=/ścieżka/do/zrzutów`. Generuje ono zrzut sterty, który można potem przeanalizować za pomocą takich narzędzi jak *jhat* lub Eclipse Memory Analyzer. Zauważ, że opcje maszyny JVM należy dodać do istniejących ustawień pamięci określonych w elemencie `mapred.child.java.opts`. Szczegółowe omówienie ustawień znajdziesz w punkcie „Ustawienia pamięci w systemie YARN i modelu MapReduce” w rozdziale 10.

### *Przeprowadzenie profilowania operacji*

Profilery Javy pozwalają dobrze przyjrzeć się pracy maszyny JVM, a Hadoop udostępnia mechanizm profilowania podzbioru operacji z zadania. Zobacz punkt „Profilowanie operacji”.

W niektórych sytuacjach warto przechowywać pliki pośrednie z nieudanych prób wykonania operacji w celu ich późniejszej analizy (zwłaszcza jeśli w katalogu roboczym operacji generowane są pomocnicze pliki zrzutu lub profilowania). Aby zachowywać takie pliki, ustaw właściwość `mapreduce.task.files.preserve.failedtasks` na wartość `true`.

Pośrednie pliki można zachowywać także dla operacji zakończonych powodzeniem. Jest to przydatne, jeśli chcesz zbadać operację, która nie powoduje awarii. Wtedy do właściwości `mapreduce.task.files.preserve.filepattern` należy przypisać wyrażenie regularne pasujące do identyfikatorów operacji, których pliki chcesz zachować.

Inną właściwością przydatną w kontekście diagnozowania jest `yarn.nodemanager.delete.debug-delay-sec`. Określa ona liczbę sekund oczekiwania na usunięcie określonych plików prób wykonania operacji (na przykład skryptu używanego do uruchamiania maszyny JVM dla kontenera operacji). Jeśli ta właściwość jest ustawiona w klastrze na wystarczająco wysoką wartość (na przykład 600, co oznacza 10 minut), programista zapewnia sobie wystarczającą ilość czasu na zapoznanie się z plikami przed ich usunięciem.

Aby zbadać pliki prób wykonania operacji, przejdź do węzła, w którym operacja zakończyła się niepowodzeniem, i poszukaj katalogu dla danej próby. Znajduje się on w jednym z lokalnych katalogów modelu MapReduce zgodnie z ustawieniami właściwości `mapreduce.cluster.local.dir` (opisanej szczegółowo w punkcie „Ważne właściwości demonów Hadoopa” w rozdziale 10.). Jeśli

do właściwości przypisana jest lista rozdzielonych przecinkami katalogów (co pozwala rozdzielić obciążenie między fizyczne dyski maszyny), znalezienie folderu dla konkretnej próby wykonania operacji może wymagać sprawdzenia wszystkich tych katalogów. Katalog próby wykonania operacji znajduje się w następującym miejscu:

```
lokalny.katalog.klastera.modelu.mapreduce/usercache/uzytkownik/appcache/  
ID-aplikacji/output/ID-próby-wykonania
```

## Dostrajanie zadania

Gdy zadanie już działa, programista może zacząć się zastanawiać, czy da się je wykonywać szybciej.

Istnieje kilku specyficznych dla Hadoopa „podejrzanych”, dla których warto sprawdzić, czy nie powodują problemów z wydajnością. Przejdź przez listę z tabeli 6.3, zanim zajmiesz się profilowaniem lub optymalizowaniem kodu na poziomie operacji.

Tabela 6.3. Lista kontrolna z obszaru dostrajania

Obszar	Zalecana praktyka	Dodatkowe informacje
Liczba mapperów	Jak długo działają mappery? Jeśli średnio jest to tylko kilka sekund, sprawdź, czy nie można używać mniejszej liczby mapperów, które pracują dłużej (około minuty). To, czy jest to możliwe, zależy od formatu danych wejściowych.	„Małe pliki i format CombineFileInputFormat” w rozdziale 8.
Liczba reduktorów	Sprawdź, czy używasz więcej niż jednego reduktora. Zgodnie z ogólną regułą operacje redukcji powinny działać około pięciu minut i generować przynajmniej blok danych.	„Określanie liczby reduktorów” w rozdziale 8.
Mechanizmy łączenia	Sprawdź, czy w zadaniu można wykorzystać mechanizm łączenia do ograniczenia ilości danych w fazie przestawiania.	„Funkcje łączące” w rozdziale 2.
Kompresja danych pośrednich	Czas wykonywania zadania prawie zawsze skraca się dzięki włączeniu kompresji danych wyjściowych z etapu mapowania.	„Kompresja danych wyjściowych z etapu mapowania” w rozdziale 5.
Niestandardowa serializacja	Jeśli używasz niestandardowych obiektów typu <code>Writable</code> lub niestandardowych komparatorów, upewnij się, że zaimplementowałeś interfejs <code>RawComparator</code> .	„Implementowanie interfejsu <code>RawComparator</code> z myślą o szybkości” w rozdziale 5.
Dostrajanie etapu przestawiania	Przestawianie w modelu MapReduce udostępnia kilkanaście parametrów służących do dostrajania zarządzania pamięcią. Mogą one pomóc w jeszcze większej poprawie wydajności.	„Dostrajanie konfiguracji” w rozdziale 7.

## Profilowanie operacji

Profilowanie (podobnie jak diagnozowanie) zadań działających w systemie rozproszonym, na przykład w modelu MapReduce, związane jest z pewnymi trudnościami. Hadoop umożliwia profilowanie niektórych operacji z zadania i po zakończeniu każdej z nich pobiera zebrane informacje na komputer programisty w celu ich późniejszych analiz z użyciem standardowych narzędzi przeznaczonych do profilowania.

Oczywiście możliwe i łatwiejsze jest profilowanie zadania uruchomionego za pomocą lokalnego mechanizmu wykonywania zadań. Jeśli masz wystarczającą ilość danych wejściowych, by zbadać operacje mapowania i redukowania, jest to wartościowy sposób na poprawę wydajności mapperów

i reduktorów. Należy jednak pamiętać o kilku zastrzeżeniach. Lokalny mechanizm wykonywania zadań to bardzo odmienne środowisko od klastra, ze zdecydowanie innymi wzorcami przepływu danych. Optymalizowanie wydajności kodu z uwzględnieniem obciążenia procesora może okazać się bezcelowe, jeśli dane zadanie w modelu MapReduce jest ograniczone wydajnością operacji wejścia-wyjścia (co jest prawdą w wielu zadaniach). Aby mieć pewność, że dostrajanie jest skuteczne, należy porównać nowy czas wykonania z dawnym w klastrze. Jednak nie jest to łatwe, ponieważ czas wykonania może się znacznie wahać z powodu współzawodnictwa o zasoby z innymi zadaniami oraz podejmowanych przez program szeregujących decyzji dotyczących lokalizacji operacji. Aby rzetelnie oszacować czas wykonywania zadania w takich warunkach, uruchom je wielokrotnie (przed zmianą i po niej) oraz sprawdź, czy poprawa jest istotna statystycznie.

Niestety, niektóre problemy (na przykład nadmierne wykorzystanie pamięci) można zreprodukować tylko w klastrze. W takich sytuacjach możliwość profilowania zadania w klastrze jest niezbędna.

## Profiler HPROF

Istnieje kilka właściwości konfiguracyjnych kontrolujących profilowanie. Można je ustawić także za pomocą metod pomocniczych z klasy `JobConf`. Aby włączyć profilowanie, wystarczy ustawić właściwość `mapreduce.task.profile` na wartość `true`.

```
% hadoop jar hadoop-examples.jar v4.MaxTemperatureDriver \  
-conf conf/hadoop-cluster.xml \  
-D mapreduce.task.profile=true \  
input/ncdc/all max-temp
```

Ta instrukcja powoduje standardowe rozpoczęcie pracy zadania, przy czym dodaje parametr `-agentlib` do polecenia Javy używanego do uruchamiania kontenerów operacji w menedżerach węzłów. Dodawany parametr można ustawić we właściwości `mapreduce.task.profile.params`. Domyślnie używany jest profiler HPROF. Jest to narzędzie do profilowania dostępne razem z platformą JDK. Jest ono proste, ale udostępnia wartościowe informacje na temat użytkowania procesora i serty przez program.

Profilowanie wszystkich operacji zadania zwykle nie ma sensu. Dlatego domyślnie uwzględniane są tylko operacje mapowania i redukcji o identyfikatorach 0, 1 i 2. Aby to zmienić, użyj właściwości `mapreduce.task.profile.maps` i `mapreduce.task.profile.reduces` do określenia zakresu identyfikatorów profilowanych operacji.

Uzyskane w trakcie profilowania każdej operacji dane wyjściowe są zapisywane razem z dziennikami w podkatalogu `userlogs` lokalnego katalogu dzienników menedżera węzła (razem z plikami `syslog`, `stdout` i `stderr`). Można je pobrać w sposób opisany w punkcie „Dzienniki w Hadoopie” i zależny od tego, czy agregacja dzienników jest włączona, czy nie.

## Przepływ pracy w modelu MapReduce

Zobaczyłeś już, jak napisać program działający w modelu MapReduce. Jednak nie opisano jeszcze, jak przekształcić problem z obszaru przetwarzania danych na postać zgodną z modelem MapReduce.

Do tego miejsca przetwarzanie danych dotyczyło stosunkowo prostego problemu — znajdowania maksymalnej temperatury zarejestrowanej w poszczególnych latach. Gdy przetwarzanie staje się bardziej skomplikowane, zwykle przejawia się to większą liczbą zadań w modelu MapReduce, a bardziej złożonymi funkcjami mapującymi lub redukującymi. Tak więc zgodnie z ogólną regułą programista powinien myśleć o dodaniu *większej liczby* zadań, a nie o dodaniu złożoności *do* zadań.

Dla bardziej skomplikowanych problemów warto rozważyć mechanizm wyższego poziomu niż model MapReduce — na przykład system Pig, Hive, Cascading, Crunch lub Spark. Oczywiście zaletą takiego podejścia jest to, że nie trzeba przekształcać problemu na zadania w modelu MapReduce. Pozwala to skoncentrować się na przeprowadzanych analizach.

Książka *Data-Intensive Text Processing with MapReduce* Jimmy'ego Lina i Chrisa Dyera (Morgan & Claypool Publishers, 2010) to doskonała pozycja pozwalająca lepiej poznać projekt algorytmu MapReduce. Gorąco zachęcamy do jej lektury.

## Rozbijanie problemu na zadania w modelu MapReduce

Przyjrzyj się teraz bardziej skomplikowanemu problemowi przekładanemu na przepływ pracy w modelu MapReduce.

Załóżmy, że programista chce znaleźć średnią maksymalną temperaturę zarejestrowaną dla każdego dnia roku w każdej stacji meteorologicznej. Aby wyznaczyć średnią maksymalną temperaturę odnotowaną w stacji 029070-999999 w dniu 1 stycznia, należy obliczyć średnią z maksymalnych temperatur zarejestrowanych w tej stacji 1 stycznia 1901 roku, 1 stycznia 1902 roku itd. aż do 1 stycznia 2000 roku.

Jak to zrobić za pomocą modelu MapReduce? Najbardziej naturalny jest podział obliczeń na dwa opisane poniżej etapy.

### 1. Obliczenie maksymalnej dziennej temperatury dla każdej pary stacja-data.

Ten program w modelu MapReduce jest odmianą programu obliczającego maksymalną temperaturę, przy czym tu klucze to pary stacja-data, a nie sam rok.

### 2. Obliczenie średniej z maksymalnych dziennych temperatur dla każdego klucza stacja-dzień-miesiąc.

Mapper przyjmuje wyjściowe rekordy z poprzedniego zadania (stacja-data, maksymalna temperatura) i przekształca je na rekordy (stacja-dzień-miesiąc, maksymalna temperatura), usuwając komponent reprezentujący rok. Funkcja redukująca określa średnią z maksymalnych temperatur dla każdego klucza stacja-dzień-miesiąc.

Otrzymane w pierwszym etapie dane wyjściowe dla analizowanych stacji wyglądają tak (skrypt `mean_max_daily_temp.sh` z przykładowym kodem zawiera implementację opartą na narzędziu Hadoop Streaming):

```
029070-99999 19010101 0
029070-99999 19020101 -94
...
```

Dwa pierwsze pola tworzą klucz. Ostatnia kolumna określa maksymalną temperaturę z wszystkich odczytów dla danej stacji i daty. Na drugim etapie dzienne maksima są uśredniane dla wszystkich lat. Oto efekt:

```
029070-99999 0101 -68
```

Oznacza to, że średnia maksymalnych dziennych temperatur 1 stycznia w stacji 029070-99999 dla ubiegłego wieku wynosi  $-6,8^{\circ}\text{C}$ .

Te obliczenia można wykonać w jednym etapie modelu MapReduce, co jednak wymaga od programisty więcej pracy<sup>2</sup>.

Argumentem na rzecz większej liczby (ale prostszych) etapów w modelu MapReduce jest to, że pozwala to budować łatwiejsze w łączeniu i konserwacji mappery oraz reduktory. W kilku studiach przypadków z części V opisano praktyczne problemy rozwiązane za pomocą modelu MapReduce. Przetwarzanie danych w każdym z tych przykładów jest zaimplementowane za pomocą dwóch lub więcej zadań w modelu MapReduce. Informacje z niniejszego rozdziału są nieocenioną pomocą, jeśli chcesz lepiej zrozumieć przekształcanie problemów na przepływ pracy w modelu MapReduce.

Funkcje mapująca i redukująca mogą być jeszcze łatwiejsze w łączeniu niż w przykładowym kodzie. Mapper często odpowiada za parsowanie danych w wejściowym formacie, projekcję (wybór odpowiednich pól) i filtrowanie (usuwanie zbędnych rekordów). W przedstawionym do tego miejsca kodzie wszystkie te czynności wykonuje jeden mapper. Można jednak rozdzielić je między odrębne mappery połączone w łańcuch za pomocą klasy bibliotecznej `ChainMapper` z Hadoopa. Dodatkowo można zastosować klasę `ChainReducer` i uruchomić w jednym zadaniu w modelu MapReduce łańcuch mapperów, po którym działa reduktor, a następnie inny łańcuch mapperów.

## JobControl

Gdy przepływie pracy w modelu MapReduce występuje więcej niż jedno zadanie, trzeba zastanowić się nad tym, jak zarządzać zadaniami, by były wykonywane w odpowiedniej kolejności. Istnieje kilka rozwiązań. Najważniejszym czynnikiem przy ich wyborze jest to, czy łańcuch zadań jest liniowy, czy tworzy bardziej skomplikowany skierowany graf acykliczny.

Gdy łańcuch jest liniowy, najprostsze podejście polega na uruchamianiu zadań jedno po drugim. Przed uruchomieniem następnego zadania należy poczekać na udane ukończenie poprzedniego.

```
JobClient.runJob(conf1);  
JobClient.runJob(conf2);
```

Jeśli zadanie zakończy się niepowodzeniem, metoda `runJob()` zgłosi wyjątek `IOException`, a dalsze zadania z potoku nie zostaną wykonane. W niektórych aplikacjach można przechwytywać ten wyjątek i zerować dane pośrednie wygenerowane przez wcześniejsze zadania.

W nowym interfejsie API modelu MapReduce rozwiązanie wygląda podobnie. Trzeba jednak sprawdzić wartość logiczną zwracaną przez metodę `waitForCompletion()` klasy `Job`. Wartość `true` oznacza, że zadanie zakończyło się powodzeniem, a wartość `false` oznacza błąd.

---

<sup>2</sup> Jest to ciekawe ćwiczenie. Oto wskazówka — wykorzystaj informacje z punktu „Sortowanie pomocnicze” w rozdziale 9.



Gdy łańcuch jest bardziej skomplikowany, można wykorzystać biblioteki pomagające w zarządzaniu przepływem pracy (działają one także dla łańcuchów liniowych, a nawet jednorazowych zadań). Najprostszym narzędziem tego typu jest klasa `JobControl` z pakietu `org.apache.hadoop.mapreduce.jobcontrol`. Dostępna jest też analogiczna klasa w pakiecie `org.apache.hadoop.mapred.jobcontrol`. Obiekt klasy `JobControl` reprezentuje graf zadań do wykonania. Należy podać konfigurację zadań, a następnie poinformować obiekt o zależnościach między zadaniami. Obiekt klasy `JobControl` działa w wątku i uruchamia zadania zgodnie z kolejnością wyznaczaną przez zależności. Można sprawdzać postęp w wykonywaniu zadań, a po ich ukończeniu — sprawdzić status każdego zadania i błędy informujące o niepowodzeniu. Jeżeli zadanie zakończy się błędem, obiekt klasy `JobControl` nie uruchomi powiązanych zadań.

## Apache Oozie

Apache Oozie to system uruchamiania przepływu pracy z powiązаныmi zadaniami. Obejmuje dwie główne części: **silnik przepływu pracy** (przechowuje i uruchamia przepływy pracy obejmujące różnego rodzaju zadania Hadoopa; mogą to być zadania z technologii MapReduce, Pig, Hive itd.) i **silnik koordynatora** (uruchamia zadania z przepływu pracy na podstawie wstępnie zdefiniowanych harmonogramów i dostępności danych). System Oozie jest zaprojektowany z myślą o skalowaniu i potrafi zarządzać wykonywaniem w klastrze Hadoopa tysięcy przepływów pracy, z których każdy może obejmować dziesiątki zadań.

Oozie ułatwia ponowne wykonywanie przepływów pracy zakończonych niepowodzeniem, ponieważ nie trzeba marnować czasu na powtarzanie poprawnie ukończonych fragmentów przepływu pracy. Każdy, kto zarządzał skomplikowanym systemem wsadowym, wie, jak trudne jest wznowianie pracy na poziomie zadań nieukończonych z powodu błędu lub przestoju maszyny. Wszystkie te osoby docenią wspomniany mechanizm. Ponadto zarządzane przez koordynatora aplikacje tworzące jeden potok danych można umieścić w *pakiecie* i uruchamiać jako połączoną jednostkę.

System Oozie (w odróżnieniu od klasy `JobControl`, która pracuje na maszynie klienckiej przesyłając zadania) działa jako usługa w klastrze. Klienci przesyłają do niej definicje przepływu pracy w celu natychmiastowego lub późniejszego uruchomienia zadań. W systemie Oozie przepływ pracy to skierowany graf acykliczny **węzłów akcji** i **węzłów sterowania przepływem**.

Węzeł akcji wykonuje operacje z przepływu pracy takie jak: przeniesienie plików w systemie HDFS, uruchomienie zadania w technologii MapReduce, Streaming, Pig lub Hive, zaimportowanie danych w narzędziu Sqoop lub uruchomienie dowolnego skryptu powłoki albo programu w Javie. Węzeł sterowania przepływem zarządza przepływem pracy pomiędzy akcjami i obsługuje logikę warunkową (pozwala to wybierać różne gałęzie programu na podstawie wyników z wcześniejszego węzła akcji) oraz równoległe wykonywanie kodu. Po zakończeniu przepływu pracy system Oozie może skierować wywołanie zwrotne HTTP do klienta, by poinformować go o statusie przepływu pracy. Ponadto można przyjmować wywołania zwrotne za każdym razem, gdy przepływ pracy wchodzi do węzła akcji lub z niego wychodzi.

## Definiowanie przepływu pracy w systemie Oozie

Definicje przepływu pracy są pisane w XML-u przy użyciu języka Hadoop Process Definition Language (jego specyfikację znajdziesz w witrynie projektu Oozie — <http://oozie.apache.org/>). Listing 6.14 przedstawia prostą definicję przepływu pracy z systemu Oozie przeznaczoną do uruchamiania jednego zadania w modelu MapReduce.

*Listing 6.14. Definicja przepływu pracy z systemu Oozie służąca do uruchamiania zadania w modelu MapReduce wyznaczającego maksymalną temperaturę*

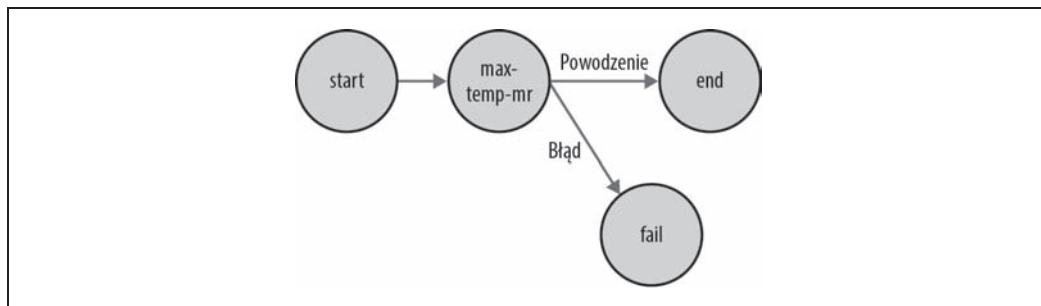
```
<workflow-app xmlns="uri:oozie:workflow:0.1" name="max-temp-workflow">
  <start to="max-temp-mr" />
  <action name="max-temp-mr">
    <map-reduce>
      <job-tracker>${resourceManager}</job-tracker>
      <name-node>${nameNode}</name-node>
      <prepare>
        <delete path="${nameNode}/user/${wf:user()}/output"/>
      </prepare>
      <configuration>
        <property>
          <name>mapred.mapper.new-api</name>
          <value>>true</value>
        </property>
        <property>
          <name>mapred.reducer.new-api</name>
          <value>>true</value>
        </property>
        <property>
          <name>mapreduce.job.map.class</name>
          <value>MaxTemperatureMapper</value>
        </property>
        <property>
          <name>mapreduce.job.combine.class</name>
          <value>MaxTemperatureReducer</value>
        </property>
        <property>
          <name>mapreduce.job.reduce.class</name>
          <value>MaxTemperatureReducer</value>
        </property>
        <property>
          <name>mapreduce.job.output.key.class</name>
          <value>org.apache.hadoop.io.Text</value>
        </property>
        <property>
          <name>mapreduce.job.output.value.class</name>
          <value>org.apache.hadoop.io.IntWritable</value>
        </property>
        <property>
          <name>mapreduce.input.fileinputformat.inputdir</name>
          <value>/user/${wf:user()}/input/ncdc/micro</value>
        </property>
        <property>
          <name>mapreduce.output.fileoutputformat.outputdir</name>
          <value>/user/${wf:user()}/output</value>
        </property>
      </configuration>
    </map-reduce>
    <ok to="end" />
    <error to="fail" />
  </action>
```

```

<kill name="fail">
  <message>Błąd zadania w modelu MapReduce.
    Komunikat o błędzie[${wf:errorMessage(wf:lastErrorNode())}]
  </message>
</kill>
<end name="end"/>
</workflow-app>

```

Ten przepływ pracy obejmuje trzy węzły sterowania przepływem (start, kill i end) i jeden węzeł akcji (map-reduce). Węzły i dozwolone przejścia między nimi są pokazane na rysunku 6.4.



Rysunek 6.4. Diagram przejść w przepływie pracy w systemie Oozie

Wszystkie przepływy pracy muszą mieć jeden węzeł start i jeden węzeł end. Gdy zadanie w przepływie pracy rozpoczyna działanie, przechodzi do węzła wskazanego w węźle start (tu węzłem docelowym jest węzeł akcji max-temp-mr). Zadanie w przepływie pracy z powodzeniem kończy działanie, gdy dochodzi do węzła end. Jeśli jednak nastąpi przejście do węzła kill, jest to uznawane za niepowodzenie. Program zwraca wtedy odpowiedni komunikat o błędzie ustawiony w elemencie message w definicji przepływu pracy.

Dużą część tego pliku definicji przepływu pracy zajmuje opis akcji map-reduce. Dwa pierwsze elementy, job-tracker i name-node, określają menedżer zasobów systemu YARN (lub jobtracker w Hadoopie 1), do którego należy przesyłać zadanie, oraz węzeł nazw (jako identyfikator URI z systemu plików Hadoopa) dla danych wejściowych i wyjściowych. Oba elementy są podawane za pomocą parametrów, dzięki czemu ta definicja przepływu pracy nie jest powiązana z żadnym konkretnym klastrzem, co pozwala na jej łatwe testowanie. Parametry są podawane jako właściwości zadania z przepływu pracy w momencie jego przesyłania, co zobaczysz dalej.



Element job-tracker ma myłą nazwę i w rzeczywistości służy do określania adresu i portu menedżera zasobów z systemu YARN.

Opcjonalny element prepare określa czynności wykonywane przed danym zadaniem w modelu MapReduce i służy na przykład do usuwania katalogów (oraz ich tworzenia, jeśli jest to konieczne, choć tu kod tego robi). Dzięki upewnieniu się przed uruchomieniem zadania, że katalog wyjściowy pozostaje w spójnym stanie, system Oozie może bezpiecznie ponownie wykonać akcję, jeśli zadanie zakończy się niepowodzeniem.

Zadanie w modelu MapReduce jest wskazywane w elemencie `configuration` przy użyciu zagnieźdżonych elementów określających pary nazwa-wartość z konfiguracji Hadoopa. Sekcję z konfiguracją modelu MapReduce możesz traktować jak deklaracyjny zastępnik klas sterowników używanych do uruchamiania programów w modelu MapReduce w innych miejscach książki (na przykład na listingu 2.5).

W kilku miejscach definicji przepływu pracy wykorzystano składnię języka JSP Expression Language (EL). System Oozie udostępnia zestaw funkcji przeznaczonych do interakcji z przepływem pracy. Na przykład funkcja `{wf:user()}` zwraca nazwę użytkownika, który uruchomił bieżące zadanie z przepływu pracy. Tu funkcja jest używana do określenia odpowiedniej ścieżki w systemie plików. W specyfikacji systemu Oozie znajdziesz listę wszystkich funkcji języka EL obsługiwanych w tym systemie.

## Pakowanie i instalowanie aplikacji opartej na przepływie pracy z systemu Oozie

Aplikacja oparta na przepływie pracy obejmuje definicję przepływu pracy oraz wszystkie powiązane zasoby (pliki JAR dla modelu MapReduce, skrypty systemu Pig itd.) potrzebne do jej wykonania. Takie aplikacje muszą mieć określoną prostą strukturę katalogów i trzeba je umieścić w systemie HDFS, tak by system Oozie miał do nich dostęp. W omawianej aplikacji wszystkie pliki znajdują się w głównym katalogu `max-temp-workflow`, co pokazano na poniższym schemacie.

```
max-temp-workflow/  
  lib/  
    hadoop-examples.jar  
  workflow.xml
```

Plik z definicją przepływu pracy, `workflow.xml`, musi znajdować się na najwyższym poziomie hierarchii. Pliki Jar z klasami aplikacji dla modelu MapReduce są umieszczone w katalogu `lib`.

Oparte na przepływie pracy aplikacje o tym układzie można zbudować za pomocą różnych narzędzi do obsługi budowania (na przykład przy użyciu Anta lub Mavena). Przykłady znajdziesz w kodzie dołączonym do tej książki. Po zbudowaniu aplikacji należy ją skopiować do systemu HDFS za pomocą standardowych narzędzi Hadoopa. Oto polecenie odpowiednie dla omawianej aplikacji:

```
% hadoop fs -put hadoop-examples/target/max-temp-workflow max-temp-workflow
```

## Uruchamianie zadań z przepływu pracy w systemie Oozie

Zobacz teraz, jak uruchomić zadanie z przepływu pracy z wczytanej właśnie do systemu aplikacji. Posłuży do tego uruchamianie z poziomu wiersza poleceń narzędzie `oozie`. Jest to program klientki przeznaczony do komunikowania się z serwerem systemu Oozie. Dla wygody eksportowana jest zmienna środowiskowa `OOZIE_URL`, informująca polecenie `oozie` o tym, który serwer systemu Oozie należy wybrać (tu używany jest serwer lokalny).

```
% export OOZIE_URL="http://localhost:11000/oozie"
```

Narzędzie `oozie` udostępnia wiele poleceń podrzędnych (ich listę zobaczysz po wpisaniu instrukcji `oozie help`). Tu używane jest polecenie podrzędne `job` z opcją `-run`, które uruchamia zadanie z przepływu pracy.

```
% oozie job -config ch06-mr-dev/src/main/resources/max-temp-workflow.properties \  
-run  
job: 0000001-140911033236814-oozie-oozi-W
```

Opcja `-config` określa lokalny plik z właściwościami Javy, w którym znajdują się definicje parametrów używanych w pliku XML z przepływem pracy (tu są to parametry `nameNode` i `resourceManager`) oraz ścieżka `oozie.wf.application.path`, informująca system Oozie o lokalizacji opartej na przepływie pracy aplikacji w systemie HDFS. Oto zawartość pliku z właściwościami.

```
nameNode=hdfs://localhost:8020
resourceManager=localhost:8032
oozie.wf.application.path=${nameNode}/user/${user.name}/max-temp-workflow
```

W celu uzyskania informacji o statusie zadania z przepływu pracy należy użyć opcji `-info` i podać identyfikator zadania wyświetlony wcześniej przez polecenie `run` (w celu wyświetlenia listy wszystkich zadań wpisz instrukcję `oozie job`).

```
% oozie job -info 0000001-140911033236814-oozie-oozi-W
```

Dane wyjściowe pokazują status zadań: `RUNNING`, `KILLED` lub `SUCCEEDED`. Wszystkie informacje można też znaleźć za pomocą sieciowego interfejsu użytkownika systemu Oozie (<http://localhost:11000/oozie>).

Po zakończeniu wykonywania zadania można sprawdzić wyniki w standardowy sposób.

```
% hadoop fs -cat output/part-*
1949 111
1950 22
```

Ten przykład pokazuje tylko niewielką część możliwości związanych z pisaniem przepływów pracy w systemie Oozie. W dokumentacji w witrynie tego systemu znajdziesz informacje o tworzeniu bardziej skomplikowanych przepływów pracy, a także o pisaniu i uruchamianiu zadań koordynatora.



## A

ACL, Access Control List, 572, 576  
ADAM, 621  
adres URL, 74  
agent platformy Flume, 366, 373  
agregacja, 526  
  danych, 475  
  dzienników, 178  
akcje, 524  
aktualizacja, 456  
  kończenie procesu, 330  
  opcjonalne finalizowanie, 330  
  opcjonalne wycofywanie, 330  
  rozpoczynanie, 329  
aktualizowanie  
  danych, 327  
  klastra, 327  
  metadanych, 327  
akumulatory, 531  
algorytm  
  PageRank, 511  
  Paxos, 583  
algorytmy  
  iteracyjne, 511  
  kompresji, 114  
analiza  
  danych, 29, 42, 44  
  genomu, 621  
  wpływu zmian parametrów, 235  
  wyrównanych odczytów, 626  
anatomia  
  odczytu pliku, 85  
  przebiegu zadania, 532  
Apache  
  Avro, 333  
  Crunch, 489

Flume, 365  
Hadoop, 645  
Hive, 445  
Mesos, 537  
Oozie, 185  
Parquet, 353  
Pig, 403  
Spark, 517  
Sqoop, 383  
API do obsługi konfiguracji, 151  
aplikacja do przesyłania zapytań, 553  
aplikacje  
  systemu YARN, 95  
  w kolejkach, 105  
arbitralne modyfikacje plików, 62  
architektura platformy Hive, 453  
ASCII, 41  
asynchroniczne wykonywanie  
  metod, 507  
atomowość, 584  
Avro, 333  
  deserializacja, 337  
  model MapReduce, 346  
  narzędzia, 342  
  pliki danych, 340  
  schematy, 343  
  schematy systemu, 334  
  serializacja, 337  
  sortowanie, 344, 349  
  typy danych, 334  
  używanie systemu, 351  
  współdziałanie języków, 341  
awaria  
  częściowa, 567  
  węzła nazw, 66

## B

bazy  
  danych, 385  
  RDBMS, 561  
bezpieczeństwo, 299  
biblioteka  
  Crunch, 490  
  libhdfs, 73  
  MRUnit, 161  
biblioteki  
  Cruncha, 513, 514  
  FlumeJava, 489  
binarne dane wejściowe, 236  
BLOB, 396  
blokada rozproszona, 594  
blokady, 544  
bloki systemu HDFS, 63, 233  
BookKeeper, 597  
budowanie aplikacji, 588  
  w modelu MapReduce, 151  
budowanie klastra, 277

## C

Capacity  
  konfigurowanie, 103  
Cascading, 629  
  elastyczność, 637  
  operacje, 632  
Crunch, 489  
  biblioteki, 513  
  funkcje, 502  
  inspekcja planu wykonania, 508  
  interfejsy API, 493  
  materializacja, 504  
  potoki, 500

- Crunch
    - wykonywanie potoku, 506
    - zapisywanie danych, 501
    - zbiory danych, 500
  - czas
    - taktu, 586
    - życia aplikacji, 97
  - częściowo ustrukturyzowane dane, 33
  - czujki, 578-580
- D**
- dane, 27
    - meteorologiczne, 653
    - nieustrukturyzowane, 33
    - pogrupowane, 609
    - pomocnicze, 268
    - relacyjne, 33
    - wejściowe, 237
      - z bazy, 238
    - wyjściowe, 45
      - binarne, 239
      - dla bazy, 244
      - tekstowe, 239
  - debugowanie zadania, 175
  - definiowanie przepływu pracy, 186
  - dekompresja
    - pliku, 116
    - strumieni, 115
  - delegowanie uprawnień, 302
  - demony
    - Hadoopa, 296, 647
    - systemu HDFS, 284
  - deserializacja, 122, 337
  - diagnozowanie, 507
  - DNA, 615
    - sekwencjonowanie, 620
    - wyrównywanie, 620
  - dodawanie
    - węzłów, 324
    - zależności, 169
  - domyślne zadanie, 220
  - dopasowywanie do wzorca, 82
  - dostęp
    - do danych, 62, 472
    - do danych kolumnowych, 623
    - do systemu HDFS, 72
  - dostrajanie
    - konfiguracji, 206
    - zadania, 181
  - DRF, Dominant Resource Fairness, 109
  - dynamiczne obiekty wywołujące, 430
  - dyrektywy pamięci podręcznej, 65
  - dystribucja, 374
    - firmy Cloudera, 651
  - działanie
    - bazy HBase, 545
    - modelu MapReduce, 191
  - dzielenie danych, 442
  - dziennik inspekcji, 315
  - dzienniki w Hadoopie, 178
- E**
- edytory kodu, 407
  - efekt stada, 595
  - eksom, 618
  - eksport, 398-401
  - elastyczność kolejek, 103
- F**
- Fair
    - konfigurowanie, 105
    - rozmieszczanie w kolejkach, 107
    - szeregowanie z opóźnieniem, 109
    - włączanie programu, 106
    - wyłączanie, 108
  - faza
    - redukcji, 204
    - scalania, 204
    - sortowania, 204
  - federacje, 65
  - fenotyp, 618
  - FIFO, 101
  - filtrowanie, 426, 434
  - Flume, 365
    - agent, 373, 376
    - dystribucja, 374
    - gwarancje dostarczenia, 373, 376
    - instalowanie platformy, 365
    - integracja z aplikacjami, 380
  - interceptory, 370
  - katalog komponentów, 381
  - konfiguracja platformy, 372, 378, 379
  - niezawodność, 368
  - partycje, 370
  - rozsyłanie danych, 372
  - transakcje, 368
  - Flume SDK, 380
  - format danych, 41, 231
    - CombineFileInputFormat, 227
    - DOT, 508
    - LobFile, 397
    - ORCFile, 147
    - Parquet, 147, 353, 356
    - RCFile, 147
    - SequenceFile, 144
    - SerDe, 471
    - Trevni, 147
  - formatowanie
    - systemu plików, 283
    - systemu plików HDFS, 649
  - formaty
    - binarne, 470
    - danych wyjściowych, 238
    - kolumnowe, 146, 353
    - kompresji, 119
    - plików, 146, 371
    - wejściowe, 222
    - wierszowe, 623
  - funkcja
    - mapująca, 44, 262
      - w Javie, 45
      - w Pythonie, 59
      - w Ruby, 57
    - redukująca, 55
      - w Javie, 46
      - w Pythonie, 60
      - w Ruby, 58
  - funkcje, 502
    - filtrujące, 423
    - łączące, 55
    - modelu MapReduce, 245
    - obliczeniowe, 423
    - UDAF, 481, 484, 486
    - UDF, 426, 430, 481
    - UDTF, 481
    - umożliwiające łączenie, 610



użytkownika, 426, 481  
wzytujące, 423  
zapisujące, 423  
FUSE, Filesystem in Userspace, 73

## G

generowanie danych, 409  
genom, 614, 619  
genomy referencyjne, 619  
grafy acykliczne, 533  
Grunt, 406  
grupa, 570  
    dołączanie członków, 573  
    tworzenie, 571  
    usuwanie, 575  
    wyświetlanie członków, 574  
grupowanie, 436  
grupy ujść, 377  
gwarancje dostarczenia danych, 376

## H

Hadoop, 25, 36, 70, 275  
    bezpieczeństwo, 299  
    budowanie klastra, 277  
    dzienniki, 178  
    instalowanie, 282, 645  
    interfejsy, 71  
    konfiguracja, 283, 285, 646  
    odczyt danych, 74  
    operacje wejścia-wyjścia, 111  
    specyfikacja klastra, 278  
    systemy plików, 71  
    testy porównawcze, 305  
    tryby, 647  
    ustawienia środowiskowe, 287  
    właściwości demonów, 289  
    właściwości konfiguracyjne, 647  
    zarządzanie konfiguracją, 286  
    zarządzanie platformą, 309  
haplotyp, 618  
HBase, 541  
    działanie, 545  
    implementacja, 544  
    instalacja, 546  
    interfejs użytkownika, 564  
    klasy, 552  
    klienty, 549

liczniki, 565  
model danych, 542  
narzędzia, 552  
skanery, 551  
system HDFS, 564  
wskaźniki, 565  
HDFS, 29, 61  
    bloki, 63  
    dostępność, 66  
    federacje, 65  
    formatowanie systemu plików,  
        283  
    integralność danych, 112  
    interfejs, 68  
    operacje, 69  
    przepływ danych, 85  
    ujścia, 369  
    uprawnienia do plików, 70  
    węzły danych, 64  
    węzły nazw, 64  
    zapisywanie bloków, 65

Hedwig, 597  
historia zadań, 173  
Hive, 394, 445  
    architektura, 453  
    format pliku, 468  
    format wierszy, 468  
    funkcje, 462  
    importowanie danych, 472  
    instalowanie platformy, 446  
    klienty, 452  
    konfigurowanie platformy, 449  
    kubelki, 464  
    operatory, 462  
    partycje, 464  
    powłoka, 446  
    system typów, 395  
    tabele, 463  
    typy danych, 458  
    typy proste, 459  
    typy złożone, 461  
    uruchamianie platformy, 449  
    usługi, 451  
HiveQL, 458  
    modyfikowanie tabel, 473  
HTTP, 72  
Hue, 452

## I

identyfikator zxid, 584  
identyfikatory zadań, 172  
IDL, interface description  
    language, 137, 621  
implementacja  
    interfejsu Writable, 132  
    interfejsu RawComparator, 135  
    MapReduce 1, 99  
    typu RecordReader, 229  
import przyrostowy, 392  
importowanie  
    danych, 385, 389, 394, 472  
    dużych obiektów, 396  
    w trybie bezpośrednim, 392  
indeksowanie, 127  
    zdarzeń, 373  
indeksy, 456  
indel, 618  
informacje  
    o plikach, 82  
    o pliku w mapperze, 228  
    o postępie i statusu, 196  
    o stanie pliku, 80  
inicjowanie zadania, 193  
inspekcja planu wykonania, 508  
instalowanie  
    Hadoopa, 282  
    HBase, 546  
    klastra, 282  
    platformy Flume, 365  
    platformy Hadoop, 645  
    platformy Hive, 446  
    Sparka, 518  
    systemu ZooKeeper, 568  
instrukcja  
    ALTER TABLE, 473  
    COGROUP, 437  
    CREATE TABLE...AS  
        SELECT, 473  
    DROP TABLE, 474  
    GROUP, 440  
    INSERT, 472  
    SET, 450  
integracja  
    danych, 111, 603, 607  
    procesorów, 603  
    semantyczna, 603

interaktywny SQL, 31  
interceptory, 370  
interfejs  
  API, 272, 273, 338  
  API dla Pythona, 341  
  API FileSystem, 75  
  API Javy, 58, 657  
  API JDBC, 389  
  CLI, 452  
  Configurable, 158  
  JDBC, 238  
  PathFilter, 83  
  RawComparator, 135  
  REST, 553  
  ResultSet, 391  
  Seekable, 77  
  systemu HDFS, 68  
  Thrift, 553  
  Tool, 158  
  użytkownika, 171  
  w Javie, 74  
  Writable, 123  
  WritableComparable, 124  
interfejsy  
  API Cruncha, 493  
  pępowinowe, 196  
intron, 618  
iteracje, 129

## J

Java, 45, 521  
jednostki zapisujące, 62  
język  
  HiveQL, 446, 458  
  IDL, 137, 605, 621  
  Pig Latin, 407, 411  
  Ruby, 57  
  Scala, 520  
  SQL, 457  
JMX, 321  
JobControl, 184  
jobtracker, 99  
JVM, Java virtual machine, 50

## K

kanał  
  oparty na pamięci, 368  
  oparty na pliku, 368

katalog, 70, 80  
Kerberos, 300  
klasa  
  ActiveKeyValueStore, 593  
klasa  
  AvroAsTextInputFormat, 351  
  AvroParquetReader, 361  
  AvroParquetWriter, 360, 361  
  AvroParquetWriter, 360  
  BlockManagerInfo, 528  
  BytesWritable, 130  
  ChainReducer, 184  
  ChecksumFileSystem, 113  
  CodecPool, 118  
  CompressionCodecFactory,  
    116, 117  
  DataByteArray, 428  
  FileInputFormat, 224, 226  
  FileOutputFormat, 240  
  FileStatus, 80  
  FileSystem, 78  
  FixedLengthInputFormat, 237  
  From, 500  
  FSDataInputStream, 76, 77  
  FSDataOutputStream, 79, 90  
  GenericOptionsParser, 158,  
    159, 272  
  GenericWritable, 131  
  InputSplit, 222  
  IntWritable, 123  
  KeyValueTextInputFormat, 233  
  LocalFileSystem, 112  
  MapDriver, 162  
  MapFile, 145  
  MapFileOutputFormat, 239  
  Mapper, 45, 46  
  MaxTemperatureDriver, 165  
  MaxWidgetId, 393, 394  
  MinimalMapReduceWith  
    ↳ Defaults, 218  
  MultipleOutputs, 241, 242, 243  
  NLineInputFormat, 234  
  NullWritable, 131  
  ObjectWritable, 131  
  ParquetOutputFormat, 358  
  ParquetWriter, 360  
  PigStorage, 433  
  PObject, 505  
  ProtoParquetWriter, 360  
  PType<S>, 497  
  klasa RDD, 521  
  Reducer, 220  
  SequenceFile, 138, 139  
  SequenceFileAsBinaryInput  
    ↳ Format, 236  
  SequenceFileAsBinaryOutput  
    ↳ Format, 239  
  SequenceFileAsTextInput  
    ↳ Format, 236  
  SequenceFileInputFormat, 236  
  SequenceFileOutputFormat, 239  
  StreamXmlRecordReader, 235  
  String, 130  
  TableInputFormat, 552  
  TableOutputFormat, 552  
  Text, 127  
  TextInputFormat, 232  
  ToolRunner, 160  
  Trash, 299  
  WholeFileInputFormat, 229  
  WholeFileRecordReader, 230  
klastry, 278  
klasy  
  biblioteczne modelu  
    MapReduce, 273  
  z rodziny InputFormat, 224  
  z rodziny OutputCommitter,  
    210  
  z rodziny OutputFormat, 238  
  z rodziny Writable, 125  
klauzula ROW FORMAT, 470, 471  
klienty, 64  
  HBase, 549  
  Hive, 452  
klucze, 45, 221  
kod genetyczny, 616  
kodek, 114  
kodeki kompresji, 115  
kodon, 616  
kodowanie struktury danych, 355  
kolejka, 103  
  danych, 88  
  potwierzeń, 88  
kolekcja  
  PCollection, 505  
  PTable, 503  
  Writable, 131  
kolumna podziału, 391

- kolumnowa baza danych, 541
  - kolumny typu BLOB, 396
  - komparatory, 124
    - niestandardowe, 136
  - komponenty platformy Flume, 381, 382
  - kompresja, 118
    - danych wyjściowych, 121
    - plików, 113
    - strumieni, 115
  - konektory Sqooqa, 385
  - konfigurowanie
    - Hadoopa, 283, 285
    - klienta ZooKeepera, 585
    - platformy Hive, 449
    - platformy Flume, 372, 378, 379
    - programu Capacity, 103
    - programu Fair, 105
    - protokołu SSH, 282
    - typów, 215, 216
    - usługi SSH, 648
    - zadania, 268
    - ZooKeepera, 599
  - konserwacja, 322
  - kontener, 96
  - kontroler przełączania awaryjnego, 67
  - kontrolowanie
    - procesu importu, 391
    - sortowania, 255
  - konwersje typów, 462
  - kopie zapasowe
    - danych, 323
    - metadanych, 322
  - kopiowanie
    - pliku, 79
    - równoległe, 91
  - kosz, 298
  - krotki, 498, 630
  - księga główna, 597
  - księgowi, 597
  - kubelki, 464, 466
  - kwalifikator rodziny kolumn, 543
- L**
- leniwe generowanie danych wyjściowych, 243
  - liczba reduktorów, 219
  - liczniki, 245, 565
    - dynamiczne, 251
    - narzędzia Streaming, 252
    - operacji, 246
    - użytkowników, 248
    - wbudowane, 245
    - zadań, 249
  - lista
    - ACL, 576, 581
    - kontroli dostępu, 572
  - lokalizacje
    - docelowe, 501
    - wyjściowe, 501
  - lokalność
    - danych, 52
    - względem procesu, 535
    - względem węzła, 535
  - lokalny magazynem metadanych, 454
- Ł**
- łańcuch zadań, 638
  - łączenie
    - danych, 442
    - danych w różnych platformach, 610
    - zasobów, 152
- M**
- macierz RAID, 279
  - magazyn metadanych, 453
  - makra, 425
  - maksymalna długość wierszy, 233
  - mapowanie, 44, 121, 202
  - mapper, 161, 266
  - MapReduce
    - budowanie aplikacji, 151
    - domyślne zadanie, 216
    - formaty, 213
    - funkcje, 245
    - inicjowanie zadania, 193
    - interfejs użytkownika, 171
    - klasy biblioteczne, 273
    - konfigurowanie typów, 215, 216
    - niepowodzenie menedżera węzła, 200
  - niepowodzenie menedżera zasobów, 201
  - niepowodzenie operacji, 198
  - niepowodzenie zarządcy aplikacji, 199
  - postęp, 196
  - przepływ pracy, 182
  - przesyłanie zadania, 192
  - przypisywanie operacji, 194
  - sortowanie, 630
  - typy, 213
  - ukończenie zadania, 197
  - wykonywanie operacji, 194, 208
  - wykonywanie zadań, 191
  - zliczanie, 630
  - maszyna wirtualna Javy, 50, 521
  - materializacja, 504
  - mechanizm
    - Java Object Serialization, 138
    - partycjonowania, 262
    - projekcji, 343
  - menedżer
    - klastra, 536
    - QJM, 67
    - węzłów, 96, 544
    - zasobów, 96, 171, 544
  - Mesos, 537
  - metadane plików, 80
  - metoda
    - amap(), 658
    - asCollection(), 506
    - await(), 572
    - cache(), 527, 528
    - combineValues(), 492, 495, 510
    - filter(), 519
    - getArgToFuncMapping(), 428
    - globStatus(), 82
    - groupByKey(), 495, 510
    - hadoopFile(), 524
    - hflush(), 90
    - hsync(), 90
    - interrupt(), 591
    - main(), 571
    - map(), 45, 526
    - parallelDo(), 493
    - parse(), 163
    - read(), 78
    - reduce(), 658
    - run(), 224, 507

metoda  
  seek(), 77  
  setJarByClass(), 48  
  setMapperClass(), 48  
  setOutputKeyClass(), 48  
  setOutputPath(), 48  
  setOutputValueClass(), 48  
  setReducerClass(), 48  
  setup(), 223  
  union(), 493  
  waitForCompletion(), 48  
  write(), 134, 593

model  
  MapReduce, 41, 45, 105, 120, 149  
  zapewniania spójności, 90, 91

modyfikowanie tabel, 473

monitorowanie, 320

## N

nakładki typu Writable, 125

narzędzia  
  bazy HBase, 552  
  systemu Avro, 342  
  unikosowe, 42  
  ZooKeepera, 575

narzędzie  
  balancer, 93, 319, 324  
  Capacity, 103, 104  
  DAG, 535  
  dfsadmin, 316  
  distcp, 91  
  Fair, 105  
  fsck, 316, 324  
  GNU Readline, 406  
  import, 387  
  spark-submit, 521  
  Streaming, 195, 209, 221, 262  
  Streaming Hadoop, 57

nauka o danych biologicznych, 613

NCDC, 41

NFS, 73

niepowodzenie  
  menedżera węzła, 200  
  menedżera zasobów, 201  
  operacji, 198  
  zarządcy aplikacji, 199

niezarządzany zarządca aplikacji, 96

normalizowanie danych, 33

numery, 577

## O

obiekt typu  
  ConfigUpdater, 590  
  Configuration, 550  
  PipeAssembly, 632

obliczanie średniej, 487

obliczeniowa funkcja UDF, 429

obserwatory, follower, 582

obsługa  
  kompresji, 117  
  konfiguracji, 151  
  nieprawidłowych danych, 177  
  serializacji, 137

odczyt danych  
  adres URL, 74  
  interfejs API FileSystem, 75  
  z plików, 140, 358

odgradzanie, 67

odmiany klasy MapFile, 146

określanie schematów, 344

Oozie, 186

opcje klasy  
  GenericOptionsParser, 160  
  ToolRunner, 160

operacje  
  mapowania, 51, 52  
  redukowania, 51  
  w systemie plików, 69  
  w usłudze ZooKeeper, 578  
  wejścia-wyjścia, 111

operacyjny przepływ danych, 605

operator, 433  
  CROSS, 439  
  FOREACH...GENERATE, 434  
  ILLUSTRATE, 409  
  JOIN, 436  
  STREAM, 435

operatory języka Pig Latin, 414

## P

pakiet Kite SDK, 611

parametry dynamiczne, 444

Parquet, 353  
  konfiguracja, 358  
  model MapReduce, 362  
  odczyt plików, 358  
  struktura pliku, 357  
  typy proste, 354  
  zapis plików, 358

parsowanie rekordów, 163

partycje, 53, 370, 464  
  zbioru RDD, 529

partycjonowanie, 262  
  danych, 240  
  poziome, 238

Pig, 403  
  bazy danych, 410  
  filtrowanie danych, 434  
  Instalowanie platformy, 404  
  podstawianie wartości, 443  
  relacje anonimowe, 443  
  sortowanie danych, 441  
  tryby wykonywania kodu, 404  
  uruchamianie platformy, 404  
  uruchamianie programów, 406  
  wczytywanie danych, 433  
  współbieżność, 442  
  zapisywanie danych, 433

Pig Latin  
  funkcje, 423  
  funkcje UDF, 426  
  instrukcje, 412, 415  
  makra, 425  
  operatory, 433  
    diagnostyczne, 414  
    relacyjne, 414  
  schematy, 419  
  struktura, 411  
  typy danych, 418  
  wyrażenia, 417

plan wykonania potoku, 509

platforma  
  Apache Hadoop, 35  
  Flume, 365  
  Hadoop, 25, 275  
  Hive, 394, 445  
  Pig, 403

plik, 70  
  configuration-1.xml, 152  
  core-site.xml, 69  
  DOT, 508

- fsimage, 311
  - hadoop-cluster.xml, 156
  - hadoop-local.xml, 156
  - POM Mavena, 154
  - tarball, 117
  - typu SequenceFile, 257
  - Widget.java, 393
  - pliki
    - bardzo duże, 61
    - binarne, 388
    - danych systemu Avro, 340, 470
    - dołączania, 325
    - dziennika edycji, 310
    - dzienników systemowych, 288
    - konfiguracyjne Hadoopa, 285
    - małe, 227
    - obrazu, 310
    - ORCFile, 470
    - Parqueta, 470
    - pomocnicze operacji, 212
    - RCFile, 470
    - SequenceFile, 470
    - tekstowe, 388
    - typu SequenceFile, 140, 231, 389, 401
  - plikowe struktury danych, 138
  - pobieranie
    - danych, 474
    - wartości liczników, 251
    - wyników, 174
  - podstawianie wartości
    - zmiennych, 153
  - podzapytania, 479
  - podział
    - danych wejściowych, 118
    - na porcje, 228
  - pojedynczy punkt krytyczny, SPOF, 66
  - pola, 630
  - polecenia
    - języka Pig Latin, 415
    - narzędzia dfsadmin, 316
    - ZooKeepera, 569
  - polecenie
    - fsck, 64
    - hadoop, 50, 59
  - polimorfizm pojedynczego nukleotydu, 618
  - POM, Project Object Model, 154
  - pomocniczy węzeł nazw, 65
  - porcje
    - danych wejściowych, 51, 233
    - zdarzeń, 369
  - POSIX, 64
  - postęp, 196
  - potok, 506, 630
    - punkty kontrolne, 512
    - uruchamianie, 506
    - zatrzymywanie, 507
  - potoki łączące pola i krotki, 630
  - powłoka
    - Grunt, 406
    - platformy Hive, 446
    - rozproszona, 98
  - poziomy utrwalania, 528
  - procedury administracyjne, 322
  - proces mysqlimport, 400
  - profiler HPROF, 182
  - profilowanie operacji, 181
  - program, *Patrz* narzędzie
  - programowanie piśmienne, 621
  - programy szeregujące, 101, 299
  - projekcja, 361
  - projekt, 331
    - Cascading, 629
    - systemu HDFS, 61
  - protokół
    - RPC, 122
    - SSH, 283, 289
  - przebieg
    - testowy, 48
    - zadania, 532
  - przechowywanie danych, 29, 468
  - przełączanie awaryjne, 67, 378
  - przepływ
    - danych, 45, 51–54, 85, 485
    - operacyjny danych, 605
    - pracy, 182, 184, 188
    - sterowania, 416
  - przestawianie, 202
  - przesyłanie
    - zadania, 192, 532
    - zapytań, 553
  - przetwarzanie
    - danych, 433
    - danych z NCDC, 654
    - dużych zbiorów danych, 614
    - iteracyjne, 31
  - pliku jako rekordu, 229
  - rozproszone, 591
  - sieciowe, 33
  - strumieni, 31
  - w zapytaniach, 30
  - wsadowe, 30
  - z udziałem ochotników, 34
  - zapytań, 413
  - przypisywanie operacji do węzłów, 194
  - pula
    - bloków, 65
    - pamięci podręcznej, 65
  - punkt kontrolny, 312, 512
  - Python, 59, 341, 522
- ## R
- RAID, 29, 279
  - ratowanie życia, 613
  - RDBMS, 561
  - redukcja, 44, 203
  - reduktor, 46, 164
  - regiony, 543
  - rejestrwanie, 451
    - dziennika inspekcji, 315
    - informacji, 320
  - rekord, 51, 222, 498
  - repliki, 89
  - reprezentacje
    - generyczne, 336
    - typów, 336
  - rodzina kolumn, 543
  - rozbijanie problemu, 183
  - rozdzielanie danych
    - pomocniczych, 268
  - rozkładanie obciążenia, 557
  - rozproszona pamięć podręczna, 269, 272
  - rozproszone
    - struktury danych, 596
    - systemy plików, 61
    - zadania, 56
  - rozsyłanie danych, 372
  - równowaga w klastrach, 92
  - równoważenie obciążenia, 380
  - RPC, remote procedure calls, 122
  - Ruby, 57

## S

Scala, 520  
scalanie  
  plików, 143  
  schematów, 422  
schemat  
  bazy danych, 386  
  nadawcy, 343  
  odbiorcy, 343  
  odczytu, 361  
sekwencje ucieczki, 399  
selektor  
  replikacji, 374  
  rozsyłania, 374  
serializacja  
  danych, 122, 137, 333, 529  
  funkcji, 502  
serwis ShareThis, 638  
sesje, 585  
silnik  
  koordynatora, 185  
  przepływu pracy, 185  
  wykonawczy, 451  
skalowanie, 51  
  systemu RDBMS, 562  
skaner bloków, 318  
skanery, 551  
skrypty modelu MapReduce, 475  
słowo kluczowe OVERWRITE, 448  
sortowanie, 202, 253, 344, 349, 475  
  częściowe, 254  
  danych, 441  
  plików, 143  
  pomocnicze, 259  
  temperatur, 260  
  wszystkich danych, 255  
  wyników, 635  
Spark, 517  
  aplikacje, 520  
  etapy, 520  
  instalowanie, 518  
  menedżery klastra, 536  
  model MapReduce, 525  
  operacje, 520  
  przebieg zadania, 532  
  wykonawcy, 536  
  YARN, 537  
  zadania, 520

  zmiennie współużytkowane, 530  
specyfikacja  
  klastra, 278  
  SAM, 621  
  systemu Avro, 333  
spekulacyjne wykonywanie  
  operacji, 209  
SPOF, 66  
spójność, 583  
  kolejności, 584  
sprawdzanie  
  aktualizacji, 330  
  poprawności, 421  
  systemu plików, 316, 324  
SQL, 457  
Sqoop, 383  
  eksport danych, 398  
  import danych, 388, 389  
  konektory, 385  
  pobieranie, 383  
  proces importu, 390  
  przyrostowy import, 392  
  transakcje, 401  
  tryb bezpośredni, 393  
Sqoop 2, 384  
stany, 587  
sterownik, 532  
  JDBC, 452  
  ODBC, 453  
strona  
  operacji, 176  
  zadania, 173, 176  
struktura  
  DNA, 615  
  katalogów węzła danych, 313  
  katalogów węzła nazw, 309  
struktury danych, 309  
strumienie, 62  
strumień  
  CompressionOutputStream, 115  
studia przypadków, 601  
sygnatury, 214  
system  
  EMPI, 608, 609  
  HDFS, 61, 290, 309, 564  
  Oozie, 188  
  RDBMS, 32  
  YARN, 51, 95, 292

systemy  
  plików w Hadoopie, 70  
  serializacji, 333, 389  
szafka, 281  
szeregowanie, 101  
operacji, 535  
  z opóźnieniem, 109

## Ś

ścieżka  
  do danych wejściowych, 224  
  do klas, 168  
ślad stosu, 321  
środowisko  
  programowania, 154  
  wykonywania operacji, 208

## T

tabela kodonów, 616  
tabele, 542  
  platformy Hive, 420, 463  
  stron internetowych, 541  
  zarządzane, 463  
  zewnętrzne, 463  
tasktracker, 99  
technika fan out, 372  
technologia  
  JMX, 321  
  Streaming, 655  
tekst z ogranicznikami, 469  
tekstowe dane wejściowe, 232  
testowanie, 48  
  mappera, 161  
  sterownika, 166  
testy  
  jednostkowe, 161, 640  
  porównawcze, 306  
  porównawcze klastra, 305  
token dostępu do bloku, 303  
tokeny, 302  
topologia sieci, 86, 280  
transakcje, 368, 401, 456  
transformacje, 524  
transkrypcja, 616  
translacja, 616

tryb  
  bezpieczny, 314  
  bezpośredni, 393  
  klastra z systemem YARN, 539  
  klienta systemu YARN, 537  
  niezależny, 582, 647  
  pseudorozproszony, 647  
  replikacji, 582  
  rozproszony, 649

tryby wykonywania kodu, 404

tworzenie  
  funkcji łączącej, 56  
  grupy, 571  
  katalogów, 80  
  katalogów użytkowników, 285  
  katalogu użytkownika, 649  
  kont użytkowników, 282  
  pakietu z zadaniem, 168  
  punktów kontrolnych, 512  
  punktu kontrolnego, 312  
  skierowanego grafu  
    acyklicznego, 533  
  węzła znode, 571  
  zbiorów RDD, 523

tymczasowe węzły znode, 577

typ  
  Flow, 634  
  PCollection<S>, 497  
  Scheme, 634  
  Tap, 634

typy  
  danych platformy Hive, 460  
  danych systemu Avro, 334  
  danych w języku IDL, 605  
  operacji, 633  
  potoków, 631  
  w modelu MapReduce, 213

## U

ujście typu hdfs, 370  
ukończenie zadania, 197  
Unicode, 128  
uprawnienia  
  do plików, 70  
  z list ACL, 582  
uruchamianie  
  demonów, 284, 649  
  interfejsu HDFS, 68

kodu, 164  
platformy Hive, 449  
potoku, 506  
programów w klastrze, 167  
systemu ZooKeeper, 568  
wykonawców Sparka, 538, 539  
zadania, 169

usługa  
  rejestrująca zdarzenia, 597  
  SSH, 648  
  ZooKeeper, 567, 576

usługi platformy Hive, 451

usprawnienia  
  w zabezpieczeniach, 303

ustawianie tożsamości  
  użytkownika, 157

ustawienia  
  pamięci, 293  
  protokołu SSH, 289  
  systemu YARN, 295

ustrukturyzowane dane, 33

usuwanie  
  danych, 84  
  grupy, 575  
  rekordów, 427  
  tabel, 474  
  węzłów, 324, 325

utrwalanie  
  danych, 527  
  partycji, 528

uwzględnianie schematu  
  przy odczycie, 456  
  przy zapisie, 456

używanie  
  schematu, 433  
  ujść, 378

## W

warstwy agentów, 374  
wartości, 221  
wartości liczników, 251  
wartość null, 421  
wczytywanie  
  danych, 430, 433, 555  
  masowe danych, 557  
wejście STDIN, 57  
wejściowe porcje danych, 222

węzeł  
  akcji, 185  
  danych, 64, 544  
  dziennika, 67  
  nadrzędny, 544  
  nazw, 64, 544  
  pasywny obserwujący, 598  
  rezerwowany, 66  
  sekwencyjny, 577  
  sterowania przepływem, 185  
  trwały, 572  
  tymczasowy, 572  
  tymczasowy znode, 577  
  znode, 570

widoki, 480

wielkość  
  bloku, 298  
  bufora, 298  
  klastra, 279  
  sterty, 288  
  strony słownika, 357

wiersz poleceń, 68

właściwości  
  demonu systemu HDFS, 292  
  demonu systemu YARN, 293  
  Hadoopa, 298  
  klasy ParquetOutputFormat, 358  
  kodeków, 117  
  kompresji, 121  
  kompresji danych  
    wyjściowych, 122  
  magazynu metadanych, 455  
  serwera HTTP, 297  
  serwera RPC, 296  
  związane z trybem  
    bezpiecznym, 314

właściwość  
  dfs.replication, 68  
  fs.defaultFS, 68

wolumin przestrzeni nazw, 65

wskaźniki, 321, 565

współbieżność, 442

współczynnik skalania, 204

współdziałanie języków, 341

wstawianie danych do wielu  
  tabel, 473

wyjątek  
  InterruptedException, 591  
  KeeperException, 591  
  SessionExpiredException, 594



wyjątki  
umożliwiające wznowienie  
pracy, 592, 596  
związane ze stanem, 592  
wykonawcy, 532, 536  
wykonywanie  
grup operacji, 579  
operacji, 194, 208, 536  
potoku, 506  
spekulacyjne, 209  
wykorzystywanie  
kompresji, 120  
wielokrotne obiektów, 503  
wyrażenia, 417  
wyszukiwanie, 31  
bloków pliku, 318  
maksymalnej temperatury,  
270, 521  
wyświetlanie  
członków grupy, 574  
list plików, 81  
zawartości plików, 143  
wyznaczanie, 108  
wyznaczanie maksymalnej  
temperatury, 490  
wyzwalacze czujek, 580  
wzorce reprezentujące pliki, 82

## X

XML, 235

## Y

YARN, 95, 537  
budowanie aplikacji, 98  
struktura działania aplikacji,  
96  
szeregowanie, 101  
żądania zasobów, 97

## Z

zadania użytkowników, 307  
zadanie, 51  
domyślne, 216  
zagregowane dane, 606  
zapis  
bloku, 65  
danych, 78, 433  
do pliku, 87, 139  
plików, 358  
zapytania  
interaktywne, 558  
o odczyty, 559  
o stacje, 558  
w systemie plików, 80  
zarządca aplikacji, 96  
zarządzanie  
blokadami, 594  
konfiguracją, 155, 588, 593  
platformą Hadoop, 309  
zatrask, latch, 572  
zatrzymywanie  
demonów, 284, 649  
potoku, 507  
zbiory  
danych  
docelowe, 500  
źródłowe, 500  
zbiory RDD, 523, 534  
zdalne  
diagnozowanie, 180  
wywołania procedur, RPC, 122  
zdalny magazyn metadanych, 455  
zestaw, ensemble, 582  
zgodność, 328  
zliczanie  
k-merów, 624  
słów, 635  
złączanie, 264  
danych, 436  
po stronie mapowania, 265  
po stronie redukcji, 265

złączenia, 476  
częściowe, 478  
na etapie mapowania, 479  
wewnętrzne, 264, 476  
zewnętrzne, 478  
zmienna  
HADOOP\_CLASSPATH, 50  
zmiennie  
rozsyłane, 531  
środowiskowe, 209  
współużytkowane, 530  
znacznik synchronizacji, 340  
znaki Unicode, 128  
ZooKeeper, 567  
aktualizacja, 584  
budowanie aplikacji, 588  
implementacja, 582  
instalowanie, 568  
interfejsy API, 579  
konfiguracja, 599  
model danych, 576  
odporność, 598  
operacje, 578  
przynależność do grupy, 570  
rozproszone struktury  
danych, 596  
sesje, 585  
spójność, 583  
stany, 587  
środowisko produkcyjne, 597  
uruchamianie, 568  
wydajność, 598  
wyjątki, 591

## Ż

źródło, 502

## Ż

żądania zasobów, 97



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄZKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

## Analiza danych z Hadoopem – i wszystko staje się prostsze!

Platforma Apache Hadoop to jedno z zaawansowanych narzędzi informatycznych. Dzięki niej można przeprowadzać różne operacje na dużych ilościach danych i znacznie skrócić czas wykonywania tych działań. Wszędzie tam, gdzie potrzebne jest szybkie sortowanie, obliczanie i archiwizowanie danych – np. w dużych międzynarodowych sklepach internetowych, serwisach społecznościowych lub wyszukiwarkach, takich jak Amazon, Facebook, Yahoo!, Apache Hadoop sprawdza się znakomicie. Jeśli potrzebne Ci narzędzie do poważnej analizy dużych zbiorów danych, nie znajdziesz lepszego rozwiązania!

Tę książkę napisał wytrawny znawca i współtwórca Hadoopa. Przedstawia w niej wszystkie istotne mechanizmy działania platformy i pokazuje, jak efektywniej używać. Dowiesz się stąd, do czego służą model MapReduce oraz systemy HDFS i YARN. Nauczysz się budować aplikacje oraz klastry. Poznasz dwa formaty danych, a także wykorzystasz narzędzia do ich pobierania i transferu. Sprawdzisz, jak wysokopoziomowe narzędzia do przetwarzania danych współdziałają z Hadoopem. Zorientujesz się, jak działa rozproszona baza danych i jak zarządzać konfiguracją w środowisku rozproszonym. Przeczytasz również o nowinkach w Hadoopie 2 i prześledzisz studia przypadków ilustrujące rolę tej platformy w systemach służby zdrowia i przy przetwarzaniu danych o genomie.

**Tom White** – jeden z czołowych ekspertów w zakresie obsługi platformy Hadoop. Członek Apache Software Foundation, inżynier oprogramowania w firmie Cloudera.

Przeczytaj i poznaj:

- Hadoop i model MapReduce
- Systemy HDFS i YARN
- Avro, Parquet, Flume i Sqoop – metody pracy z danymi
- Pig, Hive, Crunch i Spark – wysokopoziomowe narzędzia do przetwarzania danych

**Helion**

38625 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:  
● <http://helion.pl/promocje>  
Książki najczęściej czytane:  
● <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
● <http://helion.pl/nowosci>

Helion SA  
ul. Kościelna 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-1457-3



9 788328 314573

Informatyka w najlepszym wydaniu

cena: 89,00 zł