Массово-параллельный подход к обработке радарных данных

С. Е. Попов, Р. Ю. Замараев, Л. С. Миков

Институт вычислительных технологий СО РАН, Новосибирск, 630090, Россия
E-mail: popov@ict.sbras.ru

В статье описывается современный подход к созданию высокопроизводительной вычислительной системы для обработки спутниковых радарных снимков на базе технологии Apache Spark. Рассматриваются предварительные этапы полных схем обработки данных для построения скоростей смещений земной поверхности методами малых базовых линий (SBaS) и постоянных отражателей (PS). Оба метода реализуются в несколько этапов, на каждом из которых расчётный алгоритм имеет собственные настроечные параметры. Их комбинация определяет результативность отдельного этапа и всего расчёта в целом. Соответственно, возникает задача организации на массивных данных многовариантного расчёта с пользовательским контролем промежуточных результатов и подбором параметров. Для её решения разработаны адаптированные схемы автоматического запуска расчётных заданий в параллельном режиме в кластерной среде под управлением Apache Spark с использованием объектов-исполнителей. Особенностью предлагаемых решений является использование настраиваемых контейнеров-исполнителей с внутренними механизмами взаимодействия между расчётными алгоритмами и возможность объединения контейнеров в единый сценарий запуска для получения конечного решения в виде смещений поверхности. В работе приводится общее описание организации параллельных вычислений и особенности реализации конкретных этапов предварительной обработки в рамках предложенного подхода. Приведены сравнительные результаты тестирования вычислительной системы на демонстрационном кластере. Показана возможность существенного сокращения времени выполнения расчётов в задачах обработки радарных данных с использованием только открытых стандартов и свободно распространяемых библиотек программного обеспечения, а также относительно дешёвого аппаратного обеспечения.

Ключевые слова: спутниковые радарные данные, дифференциальная интерферометрия, расчёт смещений земной поверхности, массово-параллельные вычисления, Apache Spark

Одобрена к печати: 18.03.2020
DOI: 10.21046/2070-7401-2020-17-2-49-61

Введение

Основная ценность космической информации, поступающей при мониторинге земной поверхности, зачастую заключается в возможности её оперативного анализа. Поэтому активное развитие методов дифференциальной интерферометрии и средств дистанционного зондирования кроме совершенствования аппаратной части спутников также требует создания проблемно-ориентированных программных комплексов для автоматизированной и оперативной обработки большого объёма радарных данных.

Для стандартного в радарной интерферометрии аналитического аппарата DInSAR (Ferretti et al., 2006) для оценки скоростей смещений земной поверхности широкое распространение получили метод малых базовых линий (Small Baseline Subset — SBaS) (Феоктистов и др., 2015; Lanari et al., 2007) и относительно новый метод постоянных отражателей (Persistent Scatterer — PS) (Crosetto et al., 2016; Sousaa et al., 2011). Оба метода построены на совместном использовании длинных временных серий радарных изображений одной территории в повторяющейся геометрии съёмки.

Алгоритмы пред- и постобработки в полных схемах расчёта методами PS и SBaS относятся к наиболее ресурсоёмким этапам вычислений, так как содержат множественные итерации и существенно зависят в своей результативности от настроечных параметров. В обработке нередки «откаты», когда для продолжения вычислений необходимо вернуться на предыдущий этап и изменить настройки алгоритма или даже комплектность снимков. В результате полная

обработка малого по меркам радарной интерферометрии комплекта из 12 снимков размером 3000×1000 пикселей может занять 3–5 ч.

Оптимизации и ускорению обработки радарных данных, в том числе за счёт распараллеливания алгоритмов, посвящён ряд исследований. Наиболее популярной оказалась тема сокращения времени процедуры развёртки фазы с использованием графических ускорителей по технологии Nvidia CUDA (Gao et al., 2015; Guerriero et al., 2015; Karasev et al., 2007; Zhang et al., 2010).

Заметно меньше представлены работы типа (Marinkovic et al., 2004), посвящённые обработке радарных данных в многопроцессорных системах. Ещё реже встречаются такие работы, как (Costantini et al., 1999), посвящённые оптимизации самих ресурсоёмких алгоритмов.

Общим для всех работ по ускорению обработки радарных данных является то, что в них оптимизируются отдельные процедуры, а автоматизация распараллеливания вычислений практически не рассматривается. В итоге добиться значимого уменьшения общего времени расчёта от «сырых» данных до конечных значений скоростей смещений земной поверхности не удаётся.

Отсюда возникла идея поэтапного выполнения полной схемы расчёта смещений по схеме «вход – выход» в единой инфраструктуре, обеспечивающей массово-параллельные вычисления, управление ресурсами и контроль промежуточных результатов. Такая инфраструктура могла бы обеспечить высокопроизводительную обработку больших объёмов радарных данных, получаемых в потоковом режиме, и обеспечить существенное сокращение общего времени расчёта. Для реализации этой идеи выбрана технологическая платформа для массово-параллельных вычислений с открытым исходным кодом Apache Spark (https://spark.apache.org/docs/latest/).

Исходные данные

Исходными данными в алгоритмах предварительной обработки для расчёта скоростей смещений методами SBaS и PS послужили спутниковые радарные снимки аппарата Sentinel-1A. Для дифференциальной интерферометрии использовался канал типа IW (Interferometric Wide swath) с вертикальной и горизонтальной поляризацией VV+HV (https://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/). Территория охвата — Кемеровская обл., Россия. Данные были получены с открытого ресурса в сети интернет Copernicus Open Access Hub (https://scihub.copernicus.eu/dhus/#/home) Европейского космического агентства (*англ.* European Space Agency — ESA, https://sentinel.esa.int/web/sentinel/home).

Методы расчёта смещений земной поверхности

Известны и используются два метода расчёта скоростей смещений земной поверхности. Это упомянутые выше метод малых базовых линий SBaS и метод постоянных отражателей PS.

Идея метода SBaS заключается в поиске отражающих площадок, на которых влияние шумов минимально. Для таких площадок выделяются компоненты фазы на интерферограмме, обусловленные деформациями поверхности, влиянием атмосферных неоднородностей и ошибками высот на используемых цифровых моделях рельефа (ЦМР). Полагается, что атмосферные помехи в пространстве изменяются медленно, но могут быстро изменяться во времени. Из этих предпосылок вклад атмосферы и ошибок в ЦМР устанавливается и устраняется, что позволяет выделить составляющую, связанную со смещением площадок во времени.

Метод PS интересен тем, что предназначен для работы в условиях крайне низкой когерентности отражённых сигналов, из-за чего подавляющая часть элементов изображения (пикселей) оказывается неинформативной. Задачи выявления подвижек подстилающей поверхности и измерений рельефа решают с помощью этого метода для ограниченного набора постоянных отражателей. Метод позволяет решать задачи в условиях влияния атмосферных неоднородностей и высокой степени пространственной и временной декорреляции. Постоянные отражатели рассматриваются как точечные объекты типа квазиуголковых отражателей, которые обеспечивают достаточно сильный и устойчивый во времени отражённый сигнал.

Оба метода используют в совместной обработке длинные временные серии изображений одной территории, полученные в повторяющейся геометрии съёмки. В данной работе рассматриваются только предварительные этапы интерферометрической обработки: формирование интерферограмм (в том числе корегистрация) с последующей их подготовкой (удаление топографической фазы; фильтрация и коррекция относительно ЦМР) для передачи процедурам методов SBaS и StamPS. Последние процедуры являются самыми требовательными к вычислительным ресурсам.

Ниже представлены полные (стандартные) схемы расчёта смещений методами PS (*рис. 1*, см. с. 52) и SBAS (*рис. 2*, см. с. 52).

В *таблице* дано краткое описание каждой из процедур. Схемы расчёта, названия и описания процедур соответствуют программной платформе SNAP (http://step.esa.int/main/toolboxes/snap/) с компонентами Sentinel-1 Toolbox (http://step.esa.int/main/toolboxes/sentinel-1-toolbox/) от поставщика данных — Европейского космического агентства.

Описание процедур предварительной обработки радарных данных методами PS и SBaS

Название процедуры	Содержание процедуры	Количество изображений	
		на входе	на выходе
TOPS Split	Извлечение подполос (burst) из выбранной полосы (swath) съёмки в изображении	1	1
Apply-Orbit-File	Обновление векторов состояний орбит в метаданных изображения		
Back-Geocoding	Корегистрация двух изображений (главного и подчинённого) в одной и той же полосе с использованием метаданных орбит и цифровой модели рельефа (DEM)	2	
Interferogram	Вычисление свёрнутой фазы интерферограммы путём поточечного комплексного перемножения двух изображений с корректировкой на эталонную фазу с применением интерполяционного полинома 2-й степени		
TOPS Deburst	Объединение смежных подполос в направлении дальности съёмки (range) и в направлении азимута (azimuth)	1	
Topographic Phase Removal	Процедура выполняет «разглаживание» интерферограммы путём удаления топографической фазы. Процедура имитирует (формирует в оперативной памяти) интерферограмму на основе эталонной матрицы высот (DEM) и попиксельно вычитает её из исходной интерферограммы		
Goldstein Phase Filtering	Фазовая фильтрация для уменьшения фазовых остатков и повышения точности развёрнутой фазы с использованием нелинейного адаптивного алгоритма Гольдштейна (Goldstein, Werner, 1998)		
Phase Unwrapping	Двумерное развёртывание фазы — восстановление абсолютных значений фазы из двумерного массива базовых значений фазы методом потока наименьшей стоимости		
Subset	Выделение подобласти изображения		
Terrain Correction	Процедура предназначена для коррекции с использованием цифровой модели рельефа геометрических искажений (местоположения каждого пикселя) снимка. Данные искажения вызваны «боковым эффектом геометрии съёмки», появляющимся за счёт установки радиолокационного радара перпендикулярно направлению полёта. Это так называемые топографические искажения, не позволяющие корректно отобразить полученные изображения в географические системы координат, например, на электронной карте		

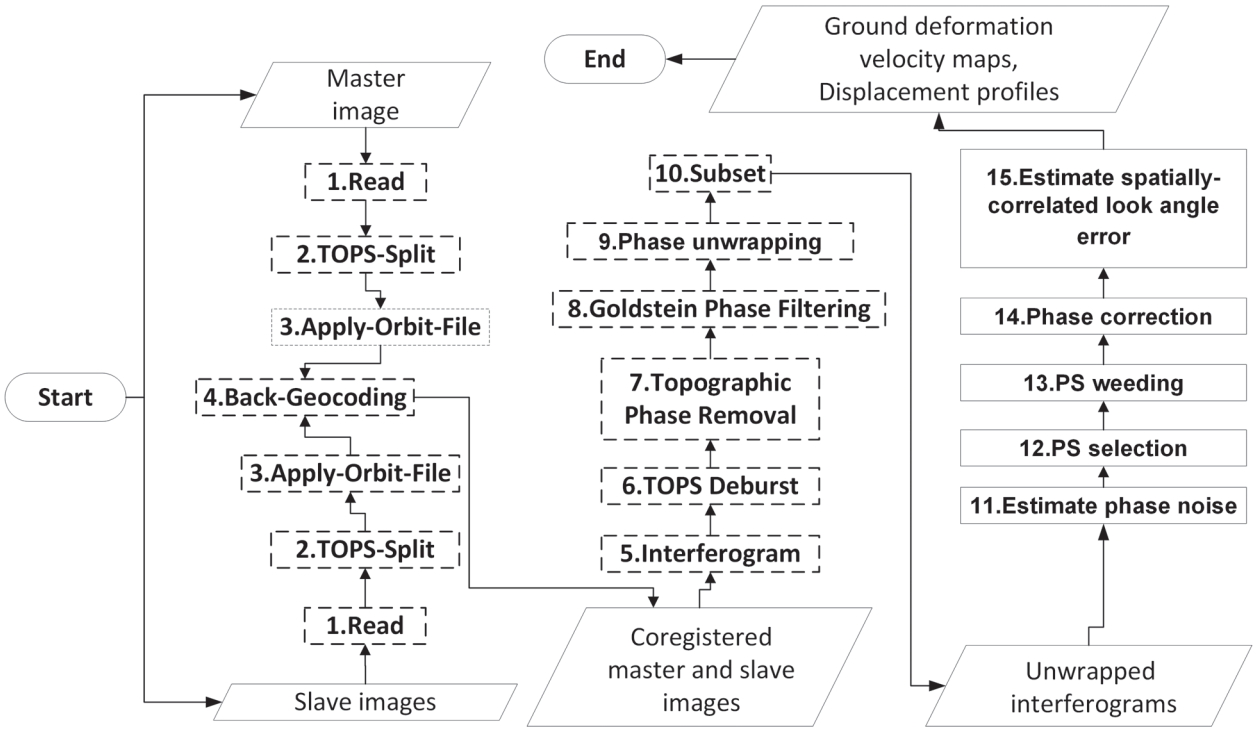

Рис. 1. Полная схема расчёта смещений методом постоянных отражателей (PS); пунктиром выделены предварительные этапы обработки

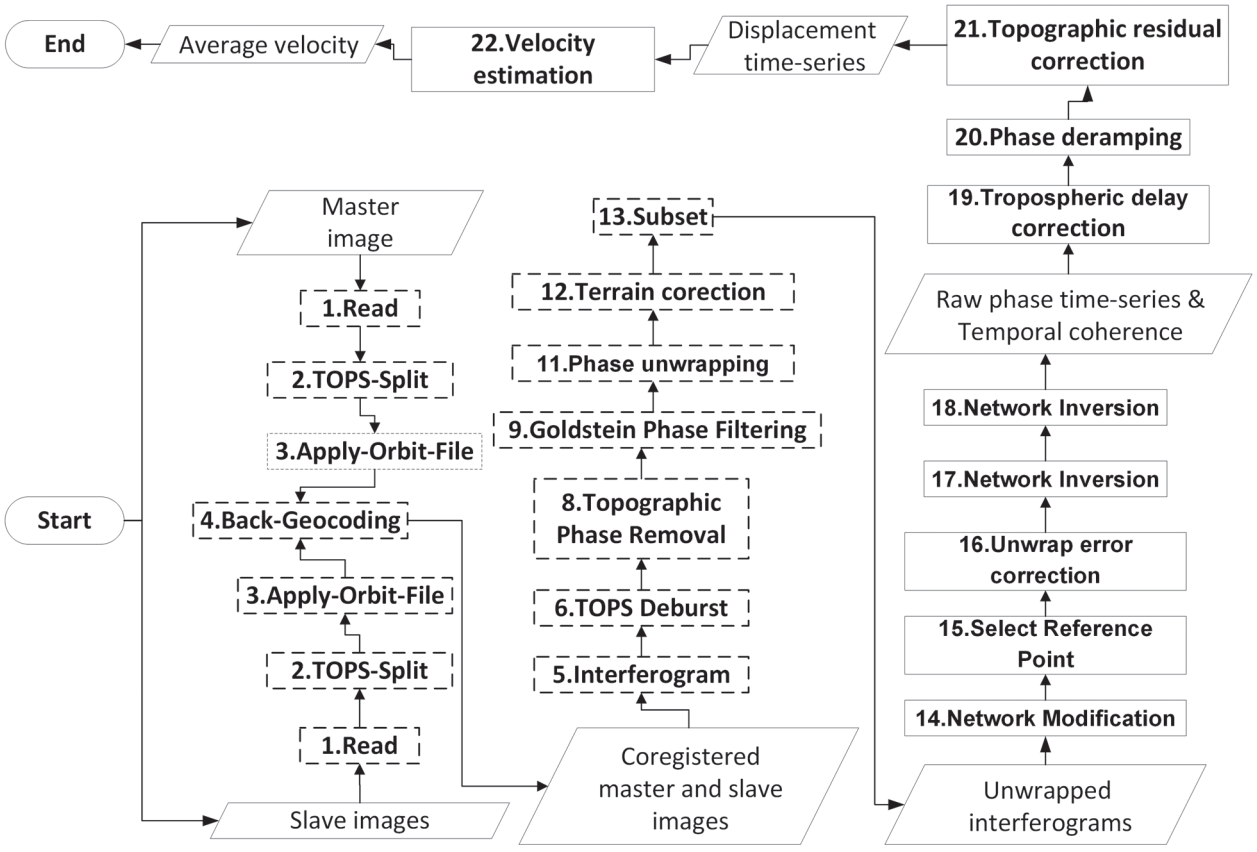

Рис. 2. Полная схема расчёта смещений методом малых базовых линий (SBAS); пунктиром выделены предварительные этапы обработки

Программно-аппаратный стек

Для решения поставленной задачи использовались следующие программные средства:

- Apache Spark API — для реализации массово-параллельного исполнения расчётов на этапах предварительной обработки, кроме развёртки фазы;
- Sentinel-1 Toolbox API — для реализации процедур предварительной обработки, кроме развёртки фазы;
- Snaphu (https://web.stanford.edu/group/radar/softwareandlinks/sw/snaphu/) — для реализации процедуры развёртки фазы (Phase unwrapping).

Для тестовых расчётов использовался вычислительный мини-кластер на базе системы управления и развёртывания компонентов Cloudera (docs.cloudera.com/index.html) со следующими техническими характеристиками: три вычислительных узла на базе процессоров Intel Core i9-9980XE (52 физических ядра, 108 потоков, RAM 376 Гбайт).

Программная реализация

Обработка данных

Программная реализация предварительных этапов обработки радарных данных в схемах расчёта смещений методами PS (см. *рис. 1*) и SBaS (см. *рис. 2*) базируется на интерфейсе программирования приложений Sentinel-1 Toolbox. Он построен на языке Java и содержит классы, реализующие функциональность каждой из процедур согласно её описанию в *таблице*. Полное описание его классов и параметров доступно по ссылке https://github.com/senbox-org/s1tbx.

Чтение входных данных (файлов снимков в формате Sentinel-1A) происходит в методе *readProduct*() класса *org.esa.snap.core.dataio.ProductIO* (см. *рис. 1–2*). На вход методу подаётся стандартный объект *java.io.File* с указанием пути к файлу снимка. Результатом работы метода является объект *org.esa.snap.core.datamodel.Product*, содержащий метаописание о загруженном снимке и значения синфазной (канал с префиксом “i”) и квадратурной (канала с префиксом “q”) компоненты сигнала для снимков Sentinel-1A/B SLC. На базе этих каналов может быть посчитана фаза как арктангенс отношения значений в каждой точке канала i и q. Эти данные далее передаются методам классов по цепочке согласно расчётным схемам.

Каждый из классов наследует абстрактный класс *OperatorSpi* и *Operator* из *org.esa.snap.core.gpf*, которые содержат методы преобразования входных данных согласно текущей процедуре. Основным методом, инициализирующим обработку, является *getTargetProduct*(). Он возвращает объект *org.esa.snap.core.datamodel.Product* с модифицированными данными и метаописанием для последующей передачи данных по расчётной схеме. Ниже представлены фрагменты кода процедур № 2–4 схем расчёта смещений (*рис. 3–5*, см. с. 54).

```
{
    String slaImglPath = "/mnt/hdfs/sladata/" +
            "S1B_IW_SLC__1SDV_20180405T002722_20180405T002752_010341_012D2E_4CAC.zip.zip";

    Product sourceProduct1 = ProductIO.readProduct(new File(slaImglPath));

    Operator op1 = (TOPSARSplitOp) (new TOPSARSplitOp.Spi().createOperator());

    // настройка параметров оператора
    ...

    Product targetProduct1 = op1.getTargetProduct();
    Operator op11 = (ApplyOrbitFileOp) (new ApplyOrbitFileOp.Spi().createOperator());
    op11.setSourceProduct(targetProduct1);
    Product targetProduct11 = op1.getTargetProduct();
}
```

Рис. 3. Фрагмент кода выполнения процедур TOPS-Split и Apply-Orbit-File схем расчёта смещений

```
{
    // параметры для TOPSARSplitOp
    op1.setParameter("selectedPolarisations", "VV");
    op1.setParameter("subswath", "IW");
    op1.setParameter("firstBurstIndex", 1);
    op1.setParameter("lastBurstIndex", 2);

    // параметры для BackGeocodingOP
    op3.setParameter("demName", "SRTM 3Sec");
    op3.setParameter("demResamplingMethod", "BICUBIC_INTERPOLATION");
    op3.setParameter("resamplingType", "BISINC_5_POINT_INTERPOLATION");
}
```

Рис. 4. Фрагмент кода параметров для оператора TOPSARSplitOp, где устанавливаются значения поляризации, интерферометрической полосы и номера подполос

```
public static void steps2_5(String resultFile){

    ...

    Operator op3 = (BackGeocodingOp) (new BackGeocodingOp.Spi().createOperator());
    op3.setSourceProducts(new Product[]{targetProduct11, targetProduct11});

    op3.setParameter("demName", "SRTM 3Sec");
    op3.setParameter("demResamplingMethod", "BICUBIC_INTERPOLATION");
    op3.setParameter("resamplingType", "BISINC_5_POINT_INTERPOLATION");

    Product targetProduct3 = op3.getTargetProduct();

    Operator op4 = (InterferogramOp) (new InterferogramOp.Spi().createOperator());
    op4.setSourceProducts(targetProduct3);

    op4.setParameter("orbitDegree", 3); // Degree of orbit (polynomial) interpolator
    op4.setParameter("srpNumberPoints", true); // Use ground square pixel
    op4.setParameter("cohWinRg", 5); // Coherence Range Window Size
    op4.setParameter("cohWinAz", 2); // Coherence Range Window Azimuth
    op4.setParameter("includeCoherence", true);

    Product targetProduct4 = op4.getTargetProduct();

    ProductIO.writeProduct(targetProduct4,
            resultFile,
            "BEAM-DIMAP");
    ...
}
```

Рис. 5. Фрагмент кода выполнения процедур Back-Geocoding и Interferogram схем расчёта смещений

Все расчётные классы содержат параметрические настройки в виде свойств, влияющих на процедуру обработки данных. Каждый параметр задаётся методом *setParameter*(*String name*, *Object value*), которому передаются название параметра и его значение.

Например, класс, реализующий корегистрацию двух изображений в одной и той же полосе, принимает на вход два объекта типа *org.esa.snap.core.datamodel.Product*. Это объекты, созданные на предыдущем шаге, главное изображение — *targetProduct11* и подчинённое изображение — *targetProduct22*. Также ему указывается метод и тип передискретизации цифровой модели рельефа.

Аналогичным способом реализуются остальные процедуры схем расчёта смещений. Отличия будут только в названиях параметров и их значениях. Промежуточные результаты выполнения процедур могут быть сохранены в распределённую файловую систему посредством метода *writeProduct*(*Product product*, *String path*, *String fileFormat*). Здесь используется встроенный формат BEAM-Dimap, содержащий метаописание промежуточного объекта *org.esa.snap.core.datamodel.Product*, а также попиксельные значения результата обработки в формате IMG с бинарной организацией данных BSQ (Band Sequental).

Параллельное выполнение

Так как обе схемы содержат одинаковые шаги предварительной обработки, за исключением процедуры Terrain correction, то далее будем рассматривать только схему для метода SBaS. На вход процедурам предварительной обработки в данных схемах подаётся пара изображений (главное и подчинённое) или интерферограмма. Каждое изображение во временном ряду, а также их попарные комбинации обрабатываются независимо на каждом шаге. Это обеспечивает возможность применения модели Map-Reduce, когда для вычисления наборов распределённых задач (шагов схемы) независимо задействуются процессорные единицы кластера. То есть для каждого изображения (пары изображений в случае процедуры Back-Geocoding) выделяется единичный процессор, память и распределённое дисковое пространство. Каждая из процедур схем может быть инкапсулирована в соответствующем java-классе и интегрирована в фреймворк Apache Spark API.

Apache Spark — это высокопроизводительная кластерная вычислительная платформа. Она является одной из реализаций модели Map-Reduce и поддерживает вычисления в общей памяти кластера, а также операции ввода-вывода в распределённой файловой системе HDFS. Одним из ключевых свойств Apache Spark является возможность работы с изолированными контейнерами-исполнителями, их автоматическое создание, управление и завершение работы в контексте параллельной обработки данных.

В нашем случае используется модифицированная модель Map-Collect, так как необходимо получить на выходе набор обработанных снимков, а не единичный результат.

Основной структурой в Spark API является гибкий распределённый набор данных (Resilient Distributed Dataset — RDD). На низкоуровневой реализации Spark API RDD является распределённой таблицей. RDD может быть полностью виртуальным либо распределённым в памяти кластера и/или в файловой системе. При создании RDD он разбивается на фрагменты (partitions) — это минимальный объём данных, который будет обработан каждым заданием в контейнере-исполнителе.

При создании RDD для последующего параллельного выполнения в него подаётся список значений типа *String*, состоящий из путей к файлам снимков (*рис. 6*). Далее создаётся объект *JavaSparkContext*, который настраивает контекст запуска контейнера-исполнителя в кластере.

```
{
    String[] slaImgPaths = new String[]{
            "/mnt/hdfs/sladata/
            S1B_IW_SLC__1SDV_20180405T002722_20180405T002752_010341_012D2E_4CAC.zip",
            "/mnt/hdfs/sladata/
            S1B_IW_SLC__1SDV_20180417T002722_20180417T002752_010516_0132C3_A59B.zip",
            // ....
            "/mnt/hdfs/sladata/
            S1B_IW_SLC__1SDV_20190412T002729_20190412T002759_015766_01D977_02DF.zip"
    };
    SparkConf sparkConf = new SparkConf();
    JavaSparkContext sc = new JavaSparkContext(sparkConf);
    List<Boolean> topsSplitResults = sc.parallelize(Arrays.asList(slaImgPaths), 32)
            .map(new InterferogramFunction("/mnt/hdfs/proc/intf/"))
            .collect();
}
```

Рис. 6. Создание и настройка среды запуска процедуры TOPS-Split в среде Apache Spark для RDD, состоящей из путей к файлам снимков

Настройка происходит при помощи объекта *SparkConf*, содержащего информацию о программе-драйвере и контейнере-исполнителе. Настройки, используемые по умолчанию, можно изменять непосредственно в коде программы либо задавать параметром *--conf* во время запуска драйвера. Настройки считываются и применяются непосредственно при выполнении кода во время инициализации объекта *SparkConf*.

Для инициализации RDD применяется метод *sc.parallelize*(*Object array*, *Integer partitions*) с передачей ему объекта *java.util.List*, содержащего значения, которые обрабатываются параллельно. Также в методе *sc.parallelize*() задаётся количество разбиений входного массива, указывающего среде Apache Spark, сколько непосредственно будет запущено параллельных заданий, равное числу путей к файлам снимков.

Метод *map*() инициализирует специальный наследуемый объект *org.apache.spark.api.java.function.Function*, который непосредственно и выполняет все действия со значениями. В нашем случае это обёртка для процедуры TOPSARSplitOp (см. *рис. 3*). Метод *collect*() собирает все результаты работы метода *map*() после завершения всех параллельных операций. В нашем случае это будет также объект *java.util.List*, но содержащий значения *true* или *false* в зависимости от успешности выполнения процедуры TOPSARSplitOp (*рис. 7*).

```
static class InterferogramFunction implements Function<String, Boolean> {
    private String pathToSaveResult;
    private String savedFileName;

    InterferogramFunction(String pathToSaveResult) {
        this.pathToSaveResult = pathToSaveResult;
    }

    @Override
    public Boolean call(String imagePath) {
        boolean result = false;
        // Формируем имя файла результата из переменной imagePath
        savedFileName = imagePath + savedFileName + ".dim";
        ...
        try {
            steps2_5(savedFileName);
            return true;
        } catch (Exception ex) {
            return false;
        }
    }
}
```

Рис. 7. Объект Function с кодом запуска процедуры TOPS-Split в параллельном режиме в среде Apache Spark API

Таким образом, появляется возможность автоматического мониторинга выполнения операторов и контроля передачи результатов далее по цепочке в расчётных схемах. Такой подход к реализации расчётной схемы позволяет автоматизировать полное её выполнение на уровне скриптов запуска отдельных процедур и/или этапов.

Отдельной задачей стало формирование гибкого файла настройки среды Apache Spark для запуска заданий (*рис. 8*).

```
spark-submit \
  --class edu.myapp.radar.processing.Interferogram \
  --master yarn \
  --deploy-mode client \
  --conf spark.executor.instances=4 \
  --conf spark.task.cpus=1 \
  --conf spark.executor.cores=8 \
  --conf spark.executor.memory="16g" \
  --conf spark.driver.memory="8g" \
  --conf spark.driver.extraClassPath="/mnt/hdfs/user/jars/spark_api/*: ...",
  --conf spark.executor.extraClassPath="/mnt/hdfs/user/jars/spark_api/*:/..." \
  ... # другие опции
  radar.processing.jar \
  .. # application-arguments
```

Рис. 8. Команда запуска программы-драйвера для исполнения процедуры TOPS-Split из схемы расчёта смещений на кластере Apache Spark

Параметры *spark.driver.extraClassPath* и *spark.executor.extraClassPath* задают пути к java-библиотекам классов и методов, которые используются в программном коде процедур расчёта смещений. Обычно задаётся общий путь как для программы-драйвера (driver), так и для контейнера-исполнителя (executor) в распределённой файловой системе HDFS. Параметры *spark.driver.memory* и *spark.executor.memory* задают выделяемую память для программы-драйвера и каждого контейнера-исполнителя соответственно. Параметр *spark.executor.cores* задаёт количество вычислительных ядер на один исполняющий контейнер от общего числа в кластере. Параметр *spark.task.cpus* задаёт количество ядер на одно задание, а *spark.executor.instances* — общее количество контейнеров-исполнителей. Запуск программы-драйвера осуществляется командой *spark-submit*.

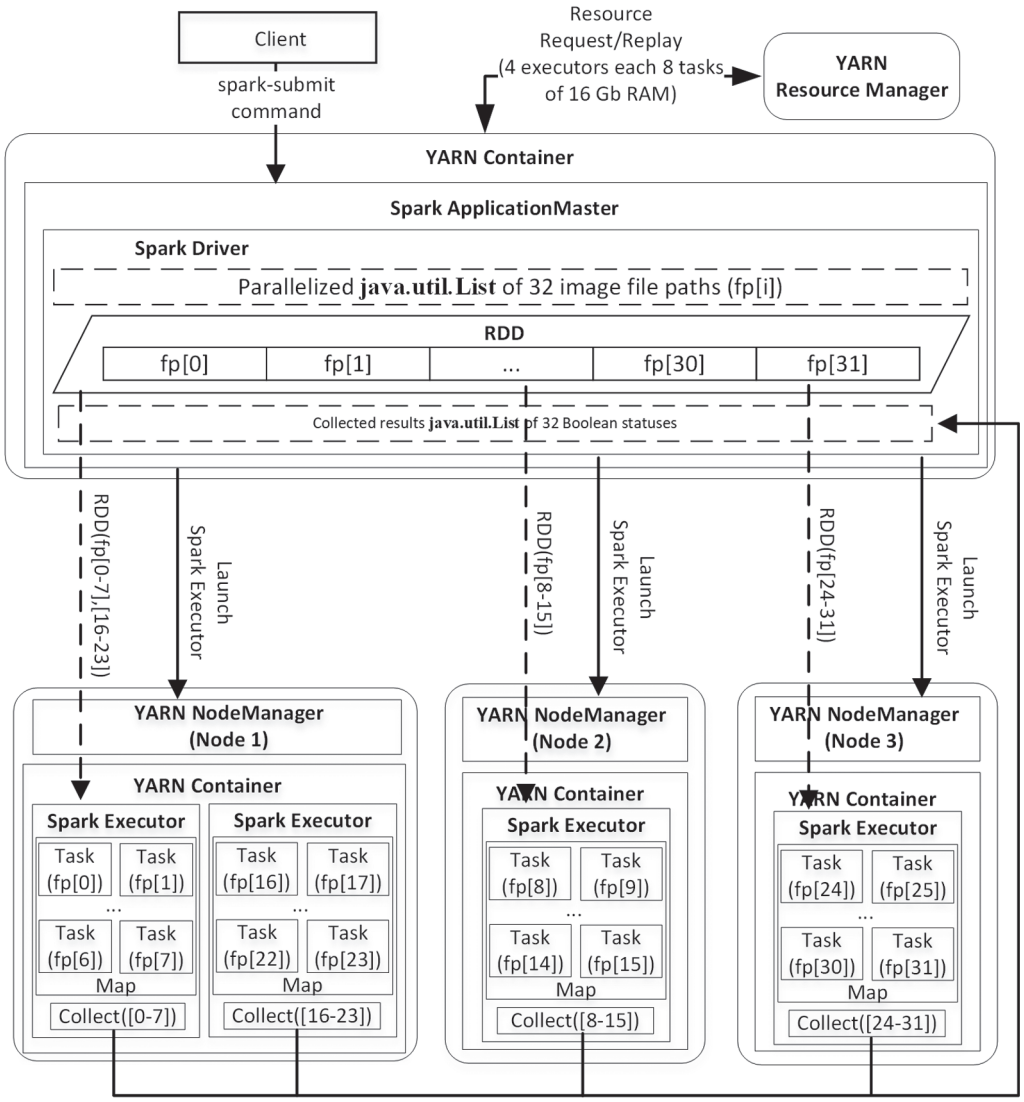

Рис. 9. Схема запуска заданий в среде Apache Spark в интеграции с менеджером управления ресурсами YARN

Параметр *master* отвечает за указание хост-узла, на котором запустится программа-драйвер и будет осуществляться управление ресурсами. В данном случае указан менеджер ресурсов YARN (http://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html). Параметр *deploy-mode* указывает тип размещения программы драйвера, т. е. запуск произойдёт на одном из узлов кластера и под неё выделятся соответствующие аппаратные и программные ресурсы согласно настройкам.

При выполнении команды конфигурации на *рис. 8* при достаточности ресурсов кластера получим: 16 вычислительных ядер и 32 Гбайт оперативной памяти, доступных на одном хост-узле, и по 8 вычислительных ядер и 16 Гбайт оперативной памяти — на двух хост-узлах. Тогда в соответствии с заданной разбивкой данных в объекте *java.util.List* (см. *рис. 6*) задания распределятся следующим образом: будет создано четыре контейнера-исполнителя по восемь заданий в каждом, при этом два контейнера разместятся на первом хост-узле и по одному — на втором и третьем хост-узлах.

Преимущество выполнения заданий на основе системы массово-параллельного исполнения Apache Spark+YARN по сравнению с Java Multi-Threading или Unix Multiple Commands Run заключается в автоматизированном управлении и распределении ресурсов с помощью нескольких компонентов (*рис. 9*, см. с. 57).

ResourceManager (RM) — менеджер ресурсов, задачей которого является распределение ресурсов, необходимых для работы приложений, и наблюдение за вычислительными узлами, на которых эти приложения выполняются.

Scheduler — планировщик, ответственный за распределение ресурсов между приложениями, которые нуждаются в ресурсах, с учётом ограничений вычислительных мощностей и наличия очереди.

ApplicationsManager (AsM) — компонент, ответственный за приём задач и запуск экземпляров ApplicationMaster, а также мониторинг узлов (контейнеров), на которых происходит выполнение. Он также предоставляет сервис для перезапуска контейнера ApplicationMaster при сбое.

ApplicationMaster (AM) — компонент, ответственный за планирование жизненного цикла, координацию и отслеживание статуса выполнения распределённого приложения. Каждое приложение имеет свой экземпляр ApplicationMaster. Он управляет всеми аспектами жизненного цикла, включая динамическое увеличение или уменьшение потребления ресурсов, управление потоком выполнения и обработку ошибок.

NodeManager (NM) — компонент, который запускается как агент (сервис) на вычислительном узле и отвечает за отслеживание используемых вычислительных ресурсов (CPU, RAM и т. д.), управление логами и отправку отчётов по используемым ресурсам планировщику менеджера ресурсов ResourceManager/Scheduler.

Весь этот инструментарий обеспечивает простую, гибкую и инкапсулированную среду выполнения параллельных заданий с развитой параметрической настройкой.

Тест производительности

Сравнение времени выполнения процедур разработанных программных компонентов проводилось с аналогичным функционалом коммерческого программного обеспечения (ПО) ENVI SARscape (https://www.harrisgeospatial.com/Software-Technology/ENVI-SARscape). Этот пакет программ позволяет обрабатывать и анализировать широкий спектр данных с существующих космических аппаратов радарной съёмки.

Тест проводился для зоны покрытия, равной единичной под-полосе (burst) одной интерферометрической полосы (IW1), поляризация вертикальная (VV). Полный тест выполнялся пять раз, при этом последовательно увеличивалось количество изображений. Выполнялась процедура Interferogram полной схемы (см. *рис. 1–2*). Начальные значения для процедуры — две пары изображений, конечное — 32 пары. На *рис. 10* (см. с. 59) приведено сравнительное время расчётов.

Программный комплекс SARScape запускался на одном из узлов кластера. Во время выполнения в параллельном режиме была задействована только часть доступных вычислительных ядер кластера (по восемь на каждый хост-узел) с целью апробации многопользовательского доступа, когда ресурсы делятся не только между заданиями одной схемы, но и между различными схемами с разными параметрическими настройками.

Указанное на *рис. 10* время выполнения не стоит рассматривать как абсолютное сравнение, так как программная часть кода SARScape является закрытой и, несомненно, отли-

чается от используемых авторами алгоритмов. Также в SARScape часть методов объединена в одну процедуру и недоступна для самостоятельного запуска. Например, формирование интерферограммы уже содержит процедуры (Apply-Orbit-File, Back-Geocoding, Goldstein Phase Filtering).

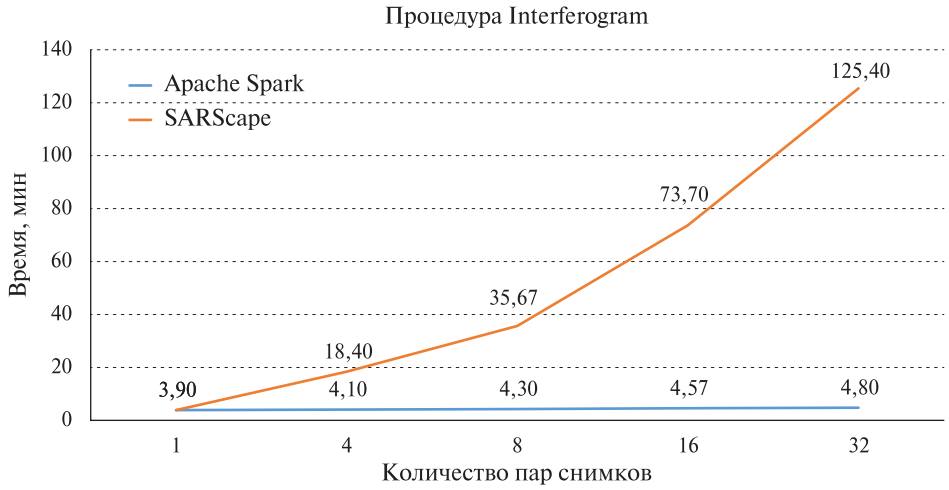

Рис. 10. Сравнительное время выполнения процедуры Interferogram в среде Apache Spark и ПО SARScape

Этим тестом авторы хотели показать возможность получения существенного выигрыша по времени при выполнении расчётов в задачах обработки радарных данных с использованием только открытых стандартов и свободно распространяемых библиотек ПО, а также относительно дешёвого аппаратного обеспечения.

Выводы

Предложенный в работе способ предварительной обработки радарных данных в системе массово-параллельного исполнения заданий Apache Spark позволяет существенно сократить время исполнения как отдельных процедур, так и полного цикла расчёта скоростей смещений. За счёт использования контейнеризации объектов-исполнителей достигнута независимость расчётов как внутри одного пула заданий, так и между различными пулами, инициализированными в многопользовательском режиме. Применение системы управления ресурсами кластера и планирования заданий YARN позволило абстрагировать все вычислительные ресурсы кластера от конкретного запуска заданий и обеспечить диспетчеризацию приложений распределённой обработки.

Использование в программном коде на основе Sentinel-1 Toolbox возможности сохранения промежуточных результатов работы процедур в схемах расчёта скоростей смещений позволяет проводить расчёты с различными параметрами, а распараллеливание обеспечивает сокращение времени расчётов по сравнению с коммерческими программными продуктами.

Использование открытого программного кода и эргономичность запуска расчётных заданий в разработанном способе обработки радарных данных обеспечивает возможность его реализации в веб-сервисах. В работе продемонстрированы приёмы адаптации программных моделей радарной интерферометрии к современным технологиям массивно-параллельных вычислений в кластерной инфраструктуре. По мнению авторов, такой подход позволит применить подобные решения и в других областях научно-технической деятельности.

Исследование выполнено при финансовой поддержке Российского фонда фундаментальных исследований и Кемеровской обл. в рамках научного проекта № 20-47-420002.

Литература

1. *Феоктистов А. А.*, *Захаров А. И.*, *Гусев М. А.*, *Денисов П. В.* Исследование возможностей метода малых базовых линий на примере модуля SBaS программного пакета SARScape и данных PCA ASAR/ENVISat и PALSAR/ALOS. Часть 1. Ключевые моменты метода // Журн. радиоэлектроники. 2015. № 9. С. 1–26.
2. *Costantini M.*, *Farina A.*, *Zirilli F.* A fast phase unwrapping algorithm for SAR interferometry // IEEE Trans. Geoscience and Remote Sensing. 1999. V. 37. No. 1. P. 452–460.
3. *Crosetto M.*, *Monserrat O.*, *Cuevas-González M.*, *Devanthéry N.*, *Crippa B.* Persistent Scatterer Interferometry: A review // ISPRS J. Photogrammetry and Remote Sensing. 2016. V. 115. P. 78–89.
4. *Ferretti A.*, *Prati C.*, *Rocca F.*, *Wasowski J.* Satellite interferometry for monitoring ground deformations in the urban environment // Proc. 10th Congress Intern. Association for Engineering Geology and the Environment (IAEG). 2006. P. 1–4.
5. *Gao Sh.*, *Zeng Q.*, *Jiao J.*, *Liang C.*, *Tong Q.* Parallel processing of InSAR interferogram filtering with CUDA programming // Science of Surveying and Mapping. 2015. No. 1. P. 54–68.
6. *Goldstein R. M.*, *Werner C. L.* Radar Interferogram Phase Filtering for Geophysical Applications // Geophysical Research Letters. 1998. V. 25. P. 4035–4038.
7. *Guerriero A.*, *Anelli V.*, *Pagliara A.*, *Nutricato R.*, *Nitti D.* Efficient implementation of InSAR time-consuming algorithm kernels on GPU environment // Proc. IEEE Intern. Geoscience and Remote Sensing Symp. (IGARSS-2015). 2015. P. 4264–4267.
8. *Karasev P.*, *Campbell D.*, *Richards M.* Obtaining a 35x Speed up in 2D Phase Unwrapping Using Commodity Graphics Processors // Proc. IEEE Radar Conf. 2007. P. 574–578.
9. *Lanari R.*, *Casu F.*, *Manzo M.*, *Zeni G.*, *Berardino P.*, *Manunta M.*, *Pepe A.* An Overview of the Small BAseline Subset Algorithm: A DInSAR Technique for Surface Deformation Analysis // Pure and Applied Geophysics. 2007. V. 164. P. 637–661.
10. *Marinkovic P. S.*, *Hanssen R. F.*, *Kampes B. M.* Utilization of Parallelization Algorithms in InSAR/PS-InSAR Processing // Proc. Envisat ERS Symp. (ESA SP-572). 2004. P. 1–7.
11. *Sousaa J. J.*, *Hooperc J. A.*, *Hanssenc R. F.*, *Bastosd L. C.*, *Ruize A. M.* Persistent Scatterer InSAR: A comparison of methodologies based on a model of temporal deformation vs. spatial correlation selection criteria // Remote Sensing of Environment. 2011. V. 115. No. 10. P. 2652–2663.
12. *Zhang F.*, *Wang B.*, *Xiang M.* Accelerating InSAR raw data simulation on GPU using CUDA // Proc. IEEE Intern. Geoscience and Remote Sensing Symp. (IGARSS-2010). 2010. P. 2932–2935.

Mass-parallel approach to radar data processing

S. E. Popov, R. Yu. Zamaraev, L. S. Mikov

Institute of Computational Technologies SB RAS, Novosibirsk 630090, Russia
E-mail: popov@ict.sbras.ru

The paper describes a modern approach to creating a high-performance computing system for processing satellite radar images based on Apache Spark technology. We consider complete data processing schemes for constructing Earth surface displacement velocities using small baseline methods (SBaS) and constant reflectors (PS). Both methods are implemented in several stages, at each of which the calculation algorithm has its own tuning parameters. Their combination determines the effectiveness of an individual stage and the entire calculation as a whole. Accordingly, the task arises of organizing on massive data a multivariate calculation with user control of intermediate results and selection of parameters. To solve it, adapted schemes for auto running computational tasks in parallel mode in a cluster environment running Apache Spark using executing objects have been developed. A feature of the proposed solutions is the use of custom containers-executors with internal mechanisms of interaction between calculation algorithms and the possibility of combining containers into a single launch scenario to obtain the final solution in the form of surface offsets. The paper provides a general description of the organization of parallel computing and describes the features of the implementation of specific stages of pre-processing in the framework of the proposed approach. Comparative results of testing the computing system on a demonstration cluster are presented. The possibility of significantly reducing

the time it takes to perform calculations in the processing of radar data using only open standards and freely distributed software libraries, as well as relatively cheap hardware, is shown.

Keywords: satellite radar data, differential interferometry, calculation of earth surface displacements, mass-parallel computing, Apache Spark

Accepted: 18.03.2020
DOI: 10.21046/2070-7401-2020-17-2-49-61

References

1. Feoktistov A. A., Zakharov A. I., Gusev M. A., Denisov P. V., Issledovanie vozmozhnostei metoda malykh bazovykh linii na primere modulya SBaS programmnogo paketa SARScape i dannykh RSA ASAR/ENVISat i PALSAR/ALOS. Chast 1. Klyuchevye momenty (Study of the possibilities of the small baseline method using the SBaS module as an example of the SARScape software package and PCA data ASAR/ENVISat and PALSAR/ALOS. Part 1. Key points of the method), *Zhurnal radioelektroniki*, 2015, No. 9, pp. 1–26.
2. Costantini M., Farina A., Zirilli F., A fast phase unwrapping algorithm for SAR interferometry, *IEEE Trans. Geoscience and Remote Sensing*, 1999, Vol. 37, No. 1, pp. 452–460.
3. Crosetto M., Monserrat O., Cuevas-González M., Devanthéry N., Crippa B., Persistent Scatterer Interferometry: A review, *ISPRS J. Photogrammetry and Remote Sensing*, 2016, Vol. 115, pp. 78–89.
4. Ferretti A., Prati C., Rocca F., Wasowski J., Satellite interferometry for monitoring ground deformations in the urban environment, *Proc. 10th Congress Intern. Association for Engineering Geology and the Environment* (*IAEG*), 2006, pp. 1–4.
5. Gao Sh., Zeng Q., Jiao J., Liang C., Tong Q., Parallel processing of InSAR interferogram filtering with CUDA programming, *Science of Surveying and Mapping*, 2015, No. 1, pp. 54–68.
6. Goldstein R. M., Werner C. L., Radar Interferogram Phase Filtering for Geophysical Applications, *Geophysical Research Letters*, 1998, Vol. 25, pp. 4035–4038.
7. Guerriero A., Anelli V., Pagliara A., Nutricato R., Nitti D., Efficient implementation of InSAR time-consuming algorithm kernels on GPU environment, *Proc. IEEE Intern. Geoscience and Remote Sensing Symp.* (*IGARSS-2015*), 2015, pp. 4264–4267.
8. Karasev P., Campbell D., Richards M., Obtaining a 35x Speed up in 2D Phase Unwrapping Using Commodity Graphics Processors, *Proc. IEEE Radar Conf.*, 2007, pp. 574–578.
9. Lanari R., Casu F., Manzo M., Zeni G., Berardino P., Manunta M., Pepe A., An Overview of the Small BAseline Subset Algorithm: A DInSAR Technique for Surface Deformation Analysis, *Pure and Applied Geophysics*, 2007, Vol. 164, pp. 637–661.
10. Marinkovic P. S., Hanssen R. F., Kampes B. M., Utilization of Parallelization Algorithms in InSAR/PS-InSAR Processing, *Proc. Envisat ERS Symp.* (*ESA SP-572*), 2004, pp. 1–7.
11. Sousaa J. J., Hooperc J. A., Hanssenc R. F., Bastosd L. C., Ruize A. M., Persistent Scatterer InSAR: A comparison of methodologies based on a model of temporal deformation vs. spatial correlation selection criteria, *Remote Sensing of Environment*, 2011, Vol. 115, No. 10, pp. 2652–2663.
12. Zhang F., Wang B., Xiang M., Accelerating InSAR raw data simulation on GPU using CUDA, *Proc. IEEE Intern. Geoscience and Remote Sensing Symp.* (*IGARSS-2010*), 2010, pp. 2932–2935.