J. Mod. Power Syst. Clean Energy (2019) 7(1):65–77
https://doi.org/10.1007/s40565-018-0406-4

Cloud-based parallel power flow calculation using resilient distributed datasets and directed acyclic graph

Dewen WANG[1], Fangfang ZHOU[1], Jiangman LI[1]

Abstract With the integration of distributed generation and the construction of cross-regional long-distance power grids, power systems become larger and more complex. They require faster computing speed and better scalability for power flow calculations to support unit dispatch. Based on the analysis of a variety of parallelization methods, this paper deploys the large-scale power flow calculation task on a cloud computing platform using resilient distributed datasets (RDDs). It optimizes a directed acyclic graph that is stored in the RDDs to solve the low performance problem of the MapReduce model. This paper constructs and simulates a power flow calculation on a large-scale power system based on standard IEEE test data. Experiments are conducted on Spark cluster which is deployed as a cloud computing platform. They show that the advantages of this method are not obvious at small scale, but the performance is superior to the stand-alone model and the MapReduce model for large-scale calculations. In addition, running time will be reduced when adding cluster nodes. Although not tested under practical conditions, this paper provides a new way of thinking about parallel power flow calculations in large-scale power systems.

CrossCheck date: 16 January 2018

Received: 22 September 2017 / Accepted: 16 January 2018 / Published online: 2 May 2018

✉ Dewen WANG
wdewen@gmail.com

Fangfang ZHOU
767334694@qq.com

Jiangman LI
lijiangman318@163.com

[1] School of Control and Computer Engineering, North China Electric Power University, Baoding 071003, China

Keywords Power flow calculation, Parallel programming model, Distributed memory-shared model, Resilient distributed datasets (RDDs), Directed acyclic graph (DAG)

1 Introduction

Power flow calculation, which refers to calculating the distribution of active power, reactive power and voltage in a power grid with given topology and component parameters, is the foundation of steady or transient state analysis, power system planning, reliability analysis and optimization. In recent years, for higher energy efficiency, lower carbon footprint, and meeting growing power demand, abundant distributed generators, storages, etc. are integrated into power grid [1], which adds more uncertainty to power flow calculation and requires it to be more scalable. Moreover, the construction of cross-regional interconnection and long-distance transmission systems make the scale of power systems expand and their structure tends to be complicated. There is an urgent need to propose new power flow calculation approaches [2, 3] to cope with these changes and consequent issues they bring to power flow calculation, such as the addition of a temporal dimension, insufficient memory, and slow computational speed and convergence [4–8].

In view of its good performance in improving computational efficiency, parallel computing is considered to be a primary solution for power flow calculation in large-scale power system. Related research mainly focuses on parallel algorithms, parallel programming models and experimental platforms. The purpose of this research is to design efficient parallel algorithms with higher parallelism and lower data correlation, using methods such as decomposition and coordination, sparse vectors, Krylov subspace iterations, and so on [9–12]. Although these methods have made some progress, it is extremely difficult to overcome the internal temporal dependencies in the process of forward-backward sweeping. They also suffer from poor versatility and portability.

The parallel programming model is closely related to the specific hardware platform. Graphical processing units (GPUs), vector machines, symmetric memory-shared multiprocessor, Transputer systems and Beowulf clusters are commonly used. Reference [10] implements power flow computing by constructing a peer-to-peer (P2P) platform, but its scalability is poor and there is a serious communication delay, so it is difficult to realize online computing of large-scale power flow on it. Reference [9] used pipeline technology with non-blocking communications based on message passing interface (MPI) to reduce waiting times in data synchronization among nodes. However, according to Amdahl's law, it has a bottleneck of gradually declining parallel speedup ratio as the number of computing nodes increases. Reference [12] proposed the parallel generalized minimal residual (GMRES) algorithm and implemented an analysis on the Beowulf cluster platform. From the experimental results, there was limited improvement in parallel efficiency, and the amount of inter-process communication was increased when adding processors. A distributed heuristic approach is proposed for the optimal management of a smart power distribution grid in [13], and the reported results show its efficiency compared to a heuristic centralized approach, however, the experiment cases are just 9-bus and 25-bus systems, which are not a practical scale. References [14] and [15] implemented parallel power flow calculations on GPUs. Reference [14] used particle swarm optimization (PSO) to calculate optimal power flow (OPF) while [15] proposed a Monte-Carlo (MC) based method for probabilistic power flow (PPF) analysis. They could get speedups of approximate 16-20 compared to sequential execution on a single CPU with 30-300 buses. These traditional parallel programming models usually have a high cost while they are difficult to implement, and their hardware platform are hard to extend. They are usually more suitable for applications with a relatively small volume of data.

Cloud computing fuses parallel computing and distributed computing and has become a standard technology for lots of big data analysis tasks including power system simulation and control. A cloud-based power system operational model called cloud grid (CG) is presented in [16], where the service model of cloud computing is integrated into the whole life cycle of the power system in China's smart grid environment. Cloud computing mainly adopts MapReduce as its parallel programming model, allowing numerous cheap commercial computers to be used for computing and giving high reliability, low cost, the capability for processing a huge amount of data, flexibility and scalability. Reference [17] designed and implemented a cloud computing platform for a wide area measurement system and it was used to process large amounts of power grid data. Its basic software is Hadoop which uses the MapReduce model for parallel data extraction, transformation and access across multiple files, demonstrating efficient massive data processing. In addition, [18] proposed a technology on processing mass intermittent energy data on MapReduce model. There are lots of advantages, achieved by allocating a large number of tasks to different nodes of a loosely-coupled computer cluster system and demanding high intensity mass data throughput; however, MapReduce-based cloud computing is most suitable for data-intensive applications, and usually has poor performance when executing computation-intensive tasks such as large-scale power flow calculations.

There has been much research devoted to improving the performance of cloud computing frameworks. Spark is considered to be a more efficient model than MapReduce. Reference [19] indicated that the bottleneck of Hadoop is reading data from disk. In [20], it was noted that Hadoop stores input and output data on disk while Spark stores them in memory, and the result showed that the latter obtained a 40 times improvement in speed over the former. Reference [21] described that, similarly to Spark's storage strategy, caching input data could halve the average job completion time. Some problems about the granularity of caching and how Hippo uses a "homework dependency graph" to determine memory retention and prefetching were discussed in [22]. As the core technique of Spark, RDDs provide a distributed memory-shared model which is characterized by MapReduce's fault-tolerance performance and simple programming. Currently, RDDs have been successfully applied in other fields requiring big data processing and analysis [23, 24] but there is little research about RDDs-based power system simulation.

In view of the above analysis, a parallel power flow calculation method derived from the Newton-Raphson method and MapReduce is proposed. The contributions of this method are mainly embodied in two aspects:

1) Deploying large-scale power flow calculation on a cloud computing platform.

2) Solving the problem of low performance of MapReduce model when applied to this task.

Considering the necessity to manage a large volume of data, the proposed method takes adequate advantage of cloud computing and makes up for the insufficiency of MapReduce by using RDDs and DAG. To reduce the time spend reading, writing and transferring data, it forms datasets in memory. The computing performance is further improved by executing the sequential programs with a maximum degree of parallelism.

2 Power flow calculation model for large-scale power systems

2.1 Power flow calculation model of power system

Power flow analysis is a fundamental calculation which is often used to evaluate the feasibility, reliability and efficiency of a power supply scheme, and in doing research on power system operation and regulating schemes [25–28]. It is also necessary for theoretical line loss analysis. An online power flow calculation with improved performance would be a significant advance.

1) Bus analysis in power flow calculation

Taking various conditions constraint into account, buses in power system are divided into different categories: ① PQ buses are the most prevalent in the power grid, with given active power and reactive power, thereby power injection is known; ② PV buses have known active power and voltage amplitudes, and usually they are few in number, if any; ③ a balance bus (PI bus) has given voltage amplitude and phase, and generally there is only one of them. A power plants buses which is responsible for adjustment of system frequency is often selected as the balance bus.

A new type of bus is appearing in the power system in increasing numbers due to the integration of distributed energy. Whether they are synchronous generators or doubly-fed generators, wind turbines can be equivalent to PQ buses in power flow calculations. If connected into grid through a current-source inverter, solar power can be equivalent to a PI bus and, if connected into grid through voltage source inverter, it can be equivalent to a PV bus. Micro gas turbines and fuel cells can be treated as PV buses in power flow calculation.

2) The mathematical model of power flow calculation

Power flow calculation aims to determine power distribution and loss accurately according to power grid parameters and structure within a reasonable time. It is necessary to set up and solve (1) Newton-Raphson method which are derived in [29].

$$\begin{bmatrix} \Delta P_1 \\ \Delta Q_1 \\ \Delta P_2 \\ \Delta Q_2 \\ \vdots \\ \Delta P_p \\ \Delta U_p^2 \\ \Delta P_n \\ \Delta U_n^2 \end{bmatrix} = \begin{bmatrix} H_{11} & N_{11} & H_{12} & N_{11} & H_{1p} & \cdots & H_{1n} \\ J_{11} & L_{11} & J_{12} & L_{12} & J_{11} & \cdots & J_{1n} \\ H_{21} & N_{21} & H_{22} & N_{22} & H_{2p} & \cdots & H_{2n} \\ J_{21} & L_{21} & J_{22} & L_{22} & J_{2p} & \cdots & J_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ H_{p1} & N_{p1} & H_{p2} & N_{p2} & H_{pp} & \cdots & H_{pn} \\ R_{p1} & S_{p1} & R_{p2} & S_{p2} & R_{pp} & \cdots & R_{pn} \\ H_{n1} & N_{n1} & H_{n2} & N_{n2} & H_{np} & \cdots & H_{nn} \\ R_{n1} & S_{n1} & R_{n2} & S_{n2} & R_{np} & \cdots & R_{nn} \end{bmatrix} \begin{bmatrix} \Delta f_1 \\ \Delta e_1 \\ \Delta f_2 \\ \Delta e_2 \\ \vdots \\ \Delta f_p \\ \Delta e_p \\ \Delta f_n \\ \Delta e_n \end{bmatrix} \tag{1}$$

where "Δ" represents the unbalance value; f_j and e_j are the real and imaginary parts of the correction voltages; H_{ij}, N_{ij}, J_{ij}, L_{ij}, R_{ij}, S_{ij} are the parts of the Jacobian matrix; P_i and Q_i are the active and reactive power; U_i is the voltage magnitude. There are $(n-1)$ PQ buses and $(n-m)$ PV buses. The first $2(m-1)$ rows in (1) correspond to the PQ buses and the following $(n-m)$ rows correspond to the PV buses. A rectangular coordinate system is used.

2.2 Power flow calculation using Newton-Raphson method

The Gauss-Sederal method, the Newton-Raphson method and the P-Q method are the main methods for power flow analysis. Meanwhile, intelligent algorithms such as the genetic algorithm and particle swarm optimization have been used; however, studies about improving power flow calculation methods are still aimed at the traditional algorithms [30, 31].

The Newton-Raphson method (or Newton method) is a typical method for solving nonlinear equations in mathematics. It is better than other methods in terms of memory requirement and computational speed with good convergence. This method is still used widely. Its basic idea is expanding the power flow equation $F(x) = 0$ as a Taylor series and omitting the second and higher order terms. This linearizes the nonlinear equations locally, and the core of the solution process is to form and solve the correction equation repeatedly.

This paper selects the Newton-Raphson method for studying power flow calculation because of the following advantages: ① Fast convergence. With an appropriate initial value, the algorithm has characteristics of square-law convergence. Generally, accurate solutions can be obtained after 4-5 iterations, and the time of iteration is substantially independent of the grid size; ② Good convergence reliability. Compared with the Gauss-Seidel method which has poor convergence and slow calculation, the Newton-Raphson method has good convergence reliability; ③ Extensive application. The Newton method is widely used and applicable to the general power flow calculation because it has no specific requirements on the parameters in power grids; ④ Easy to parallelize. Traditional direct methods for power flow calculation require forward-backward sweeping in the solving process and it is difficult to realize this as a vectorized or parallel calculation. The Newton-Raphson method is easier to parallelize.

3 Parallel power flow calculation method based on RDDs and DAG

3.1 Parallel programming model: MapReduce

MapReduce, a parallel programming model for processing large data sets, greatly simplifies the complexity of distributed parallel programming. The Map (mapping) function is used to map a set of key-value pairs to a new set of key-value pairs and The Reduce (reduction) function performs a summary operation for each group of mapped key-value pairs which share the same key. Taking the calculation of the admittance matrix's diagonal elements as an example, Fig. 1 explains the MapReduce operation.

Figure 1 shows that it is necessary to read power grid parameters from different buses before the Map operation (each rectangle represents a node), and buffer the intermediate results to disks at various nodes. During the Reduce operation, it is necessary to read intermediate results from all child nodes. As a result, there are relatively many data reading, writing and transmission operations. A growing amount of power data and calculation makes transmission delays more and more obvious. Reference [32] applies MapReduce programming to parallel power flow computing using the component averaging (CAV) iterative method and Jacobi iterative method. However, there are still problems like data transmission delays in the computing process, long processing time, and poor real-time performance, which make the results not so satisfactory.

3.2 Distributed memory sharing model

RDDs and DAG optimization are the core technologies of Spark. As a distributed memory sharing model, RDDs can store data in memory and make them form datasets. It can reduce the time spent reading and writing data by directly visiting these datasets when analyzing the data with Spark. Before calculating, the Spark kernel will draw a DAG describing the calculation path. Computation will be divided according to the DAG into sets of tasks, which will be submitted to the compute nodes for real calculations. DAG optimization enables the sequential tasks to be executed in parallel to the greatest possible extent. In addition, transmission time and execution time are reduced by implementing the task sets with narrow dependence in one stage. Figure 2 shows the calculation process of admittance matrix's diagonal elements based on RDDs and DAG optimization.

It can be seen from Fig. 2 that the branch admittance value slices are first read from each node (each rectangle represents a node), and then the intermediate key values are stored in the RDDs collection after the Map operation. The intermediate key values are read from all nodes when performing a Reduce operation. Reading and writing operations in the whole process are based on RDDs datasets and their speed is much faster. DAG optimization takes

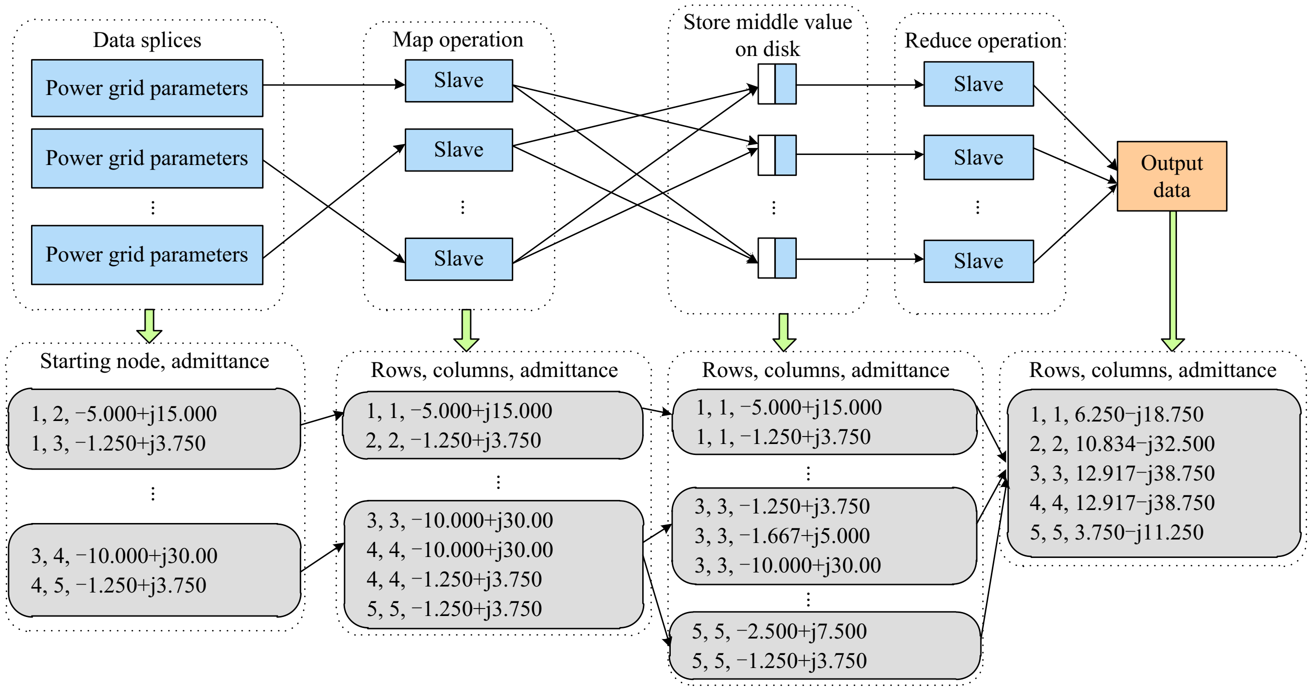

Fig. 1 Calculation process of admittance matrix's diagonal elements based on MapReduce

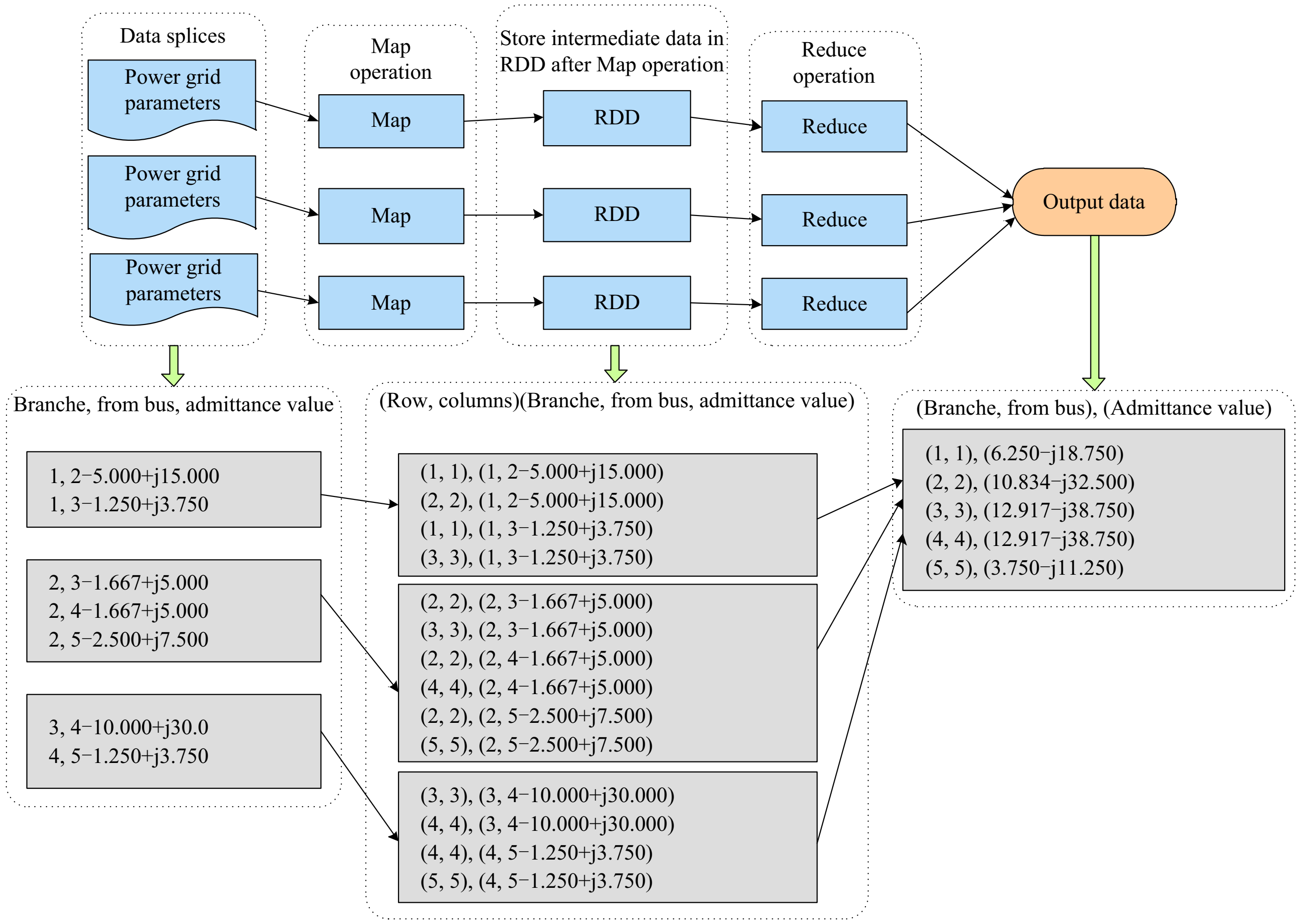
Data splices
Power grid parameters
Power grid parameters
Power grid parameters
Map operation
Map
Map
Map
Store intermediate data in RDD after Map operation
RDD
RDD
RDD
Reduce operation
Reduce
Reduce
Reduce
Output data
Branche, from bus, admittance value
1, 2−5.000+j15.000
1, 3−1.250+j3.750
2, 3−1.667+j5.000
2, 4−1.667+j5.000
2, 5−2.500+j7.500
3, 4−10.000+j30.0
4, 5−1.250+j3.750
(Row, columns)(Branche, from bus, admittance value)
(1, 1), (1, 2−5.000+j15.000)
(2, 2), (1, 2−5.000+j15.000)
(1, 1), (1, 3−1.250+j3.750)
(3, 3), (1, 3−1.250+j3.750)
(2, 2), (2, 3−1.667+j5.000)
(3, 3), (2, 3−1.667+j5.000)
(2, 2), (2, 4−1.667+j5.000)
(4, 4), (2, 4−1.667+j5.000)
(2, 2), (2, 5−2.500+j7.500)
(5, 5), (2, 5−2.500+j7.500)
(3, 3), (3, 4−10.000+j30.000)
(4, 4), (3, 4−10.000+j30.000)
(4, 4), (4, 5−1.250+j3.750)
(5, 5), (4, 5−1.250+j3.750)
(Branche, from bus), (Admittance value)
(1, 1), (6.250−j18.750)
(2, 2), (10.834−j32.500)
(3, 3), (12.917−j38.750)
(4, 4), (12.917−j38.750)
(5, 5), (3.750−j11.250)

Fig. 2 Calculation process of admittance matrix's diagonal elements based on RDDs and DAG optimization

the key values of admittance data slices into a joint partition and stores intermediate results, which are within the same key, in the same node for processing. This will avoid the data being mixed among nets, which can reduce data transmission and improve efficiency of the computation.

The difficulty of parallel processing depends on the size of the power grid and the complexity of its structure, and on how well the algorithm can be adapted to parallel computation. Several factors such as the method of subdivision, optimal allocation across nodes, and interdependency should be considered to implement the parallel power flow calculation. Figure 2 shows the principle for calculating diagonal elements of the admittance matrix based on MapReduce, including some steps which are indicated diagrammatically such as the optimal allocation of tasks and node calling.

3.3 Parallel power flow calculation method

The analysis above explains why a parallel power flow calculation using the Newton-Raphson method with RDDs and DAG optimization should be efficient. In large-scale power system, the amount of calculation for power flow calculation mainly comes from solving correction equations and updating the Jacobian matrix. The complete process includes preparation for these steps and is outlined in Fig. 3.

In the calculation process, the n buses of the power grid are not divided sequentially into PV buses, PQ buses, and balance buses. Rectangular coordinate is selected for representation of voltage. The source and intermediate data are stored in the RDDs array. After DAG optimization, those operations with narrow dependence are implemented in one stage. The whole procedure is divided into seven stages and each stage can be completed at one node independently. Different stages, however, demand striding several nodes. Stages 1, 2, and 3 can be executed in parallel as can Stages 4 and 5.

The specific procedures of each stage are as follows, where the notation $**[][]$ on $**[]$ expresses a variable array or a variable which is formed in the process of the power flow calculation ("$*$" indicates one or more letters).

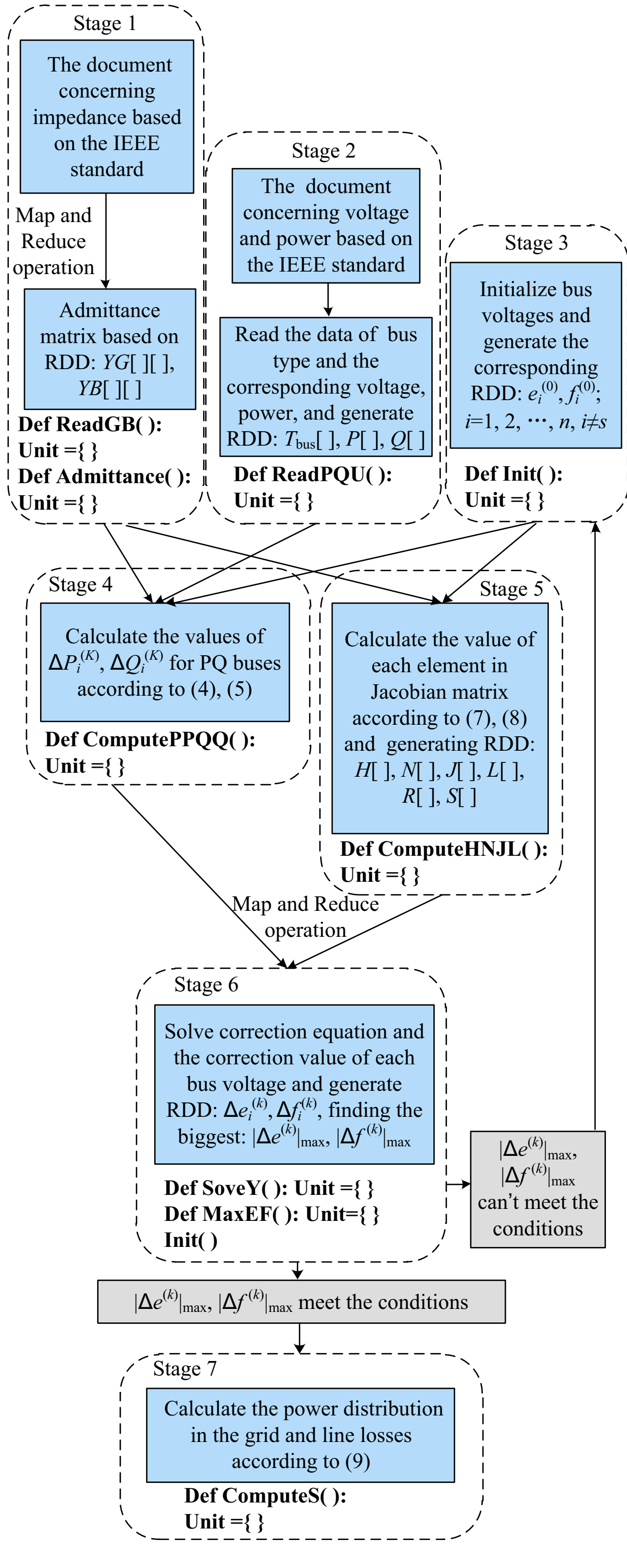
Stage 1
The document concerning impedance based on the IEEE standard
Map and Reduce operation
Admittance matrix based on RDD: YG[ ][ ], YB[ ][ ]
Def ReadGB( ): Unit ={ }
Def Admittance( ): Unit ={ }
Stage 2
The document concerning voltage and power based on the IEEE standard
Read the data of bus type and the corresponding voltage, power, and generate RDD: Tbus[ ], P[ ], Q[ ]
Def ReadPQU( ): Unit ={ }
Stage 3
Initialize bus voltages and generate the corresponding RDD: e_i^(0), f_i^(0); i=1, 2, ⋯, n, i≠s
Def Init( ): Unit ={ }
Stage 4
Calculate the values of ΔP_i^(K), ΔQ_i^(K) for PQ buses according to (4), (5)
Def ComputePPQQ( ): Unit ={ }
Stage 5
Calculate the value of each element in Jacobian matrix according to (7), (8) and generating RDD: H[ ], N[ ], J[ ], L[ ], R[ ], S[ ]
Def ComputeHNJL( ): Unit ={ }
Map and Reduce operation
Stage 6
Solve correction equation and the correction value of each bus voltage and generate RDD: Δe_i^(k), Δf_i^(k), finding the biggest: |Δe^(k)|max, |Δf^(k)|max
Def SoveY( ): Unit ={ }
Def MaxEF( ): Unit={ }
Init( )
|Δe^(k)|max, |Δf^(k)|max can't meet the conditions
|Δe^(k)|max, |Δf^(k)|max meet the conditions
Stage 7
Calculate the power distribution in the grid and line losses according to (9)
Def ComputeS( ): Unit ={ }

Fig. 3 Complete process for parallel power flow calculation

Step 1: Save the impedance data and their row numbers in the source file in a RDDs array $RR[l_{num}][4]$ and index variable l_{num} respectively. The other elements of the array are the identities of "from" and "to" buses of the branch impedances, stored in $RR[\,][0]$ and $RR[\,][1]$ respectively, and the resistance and reactance of the branch impedances, stored in $RR[\,][2]$ and $RR[\,][3]$ respectively, with ohms as their units. The conductance (real part) $YG[n+1][n+1]$ and susceptance (imaginary part) $YB[n+1][n+1]$ of the admittance matrix can be calculated according to (2) and (3) after getting array $RR[\,][\,]$. $YG[\,][\,]$ and $YB[\,][\,]$ are both RDDs arrays.

$$\begin{cases} YG[i][j] = -RR[\,][2]/(RR[\,][2]^2 + RR[\,][3]^2) \\ YB[i][j] = -RR[\,][3]/(RR[\,][2]^2 + RR[\,][3]^2) \end{cases} \tag{2}$$

$$\begin{cases} YG[i][j] = -\sum\limits_{\substack{k=1 \\ k!=i}}^{k=n} YG_{(i,k)} \\ YB[i][j] = -\sum\limits_{\substack{k=1 \\ k!=i}}^{k=n} YB_{(i,k)} \end{cases} \tag{3}$$

This procedure is shown in Fig. 4 below in terms of diagonal elements and off-diagonal elements separately. Off-diagonal elements can be computed directly from branch resistance and impedance using (2). Then the diagonal elements can be calculated according to (3).

Step 2: Read the information about bus voltage and bus power, and save the buses' type, voltage, active power and reactive power in the RDDs arrays $T_{bus}[n+1]$, $U[n+1]$, $P[n+1]$ and $Q[n+1]$, respectively. Set the values of unknown data to zero; for instance, the reactive power of a PV bus is unknown, so these are set zero.

Step 3: Set the initial value of voltage on each bus according to (4). Each voltage value consists of real part e and imaginary part f, while their correction values are ee and ff. In the first iteration, initial values and correction

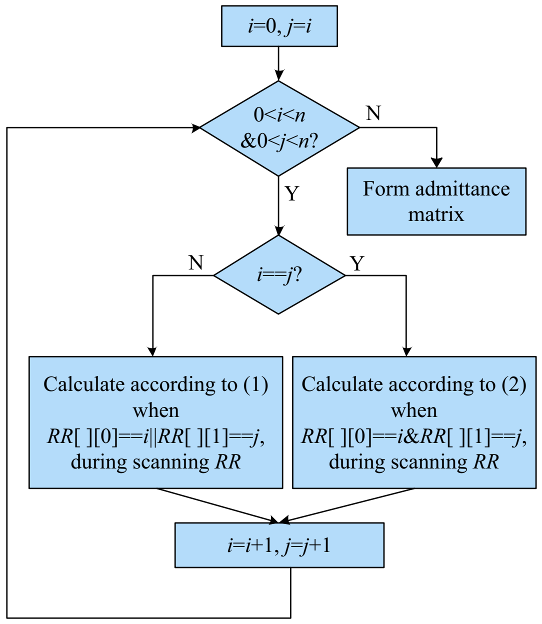
i=0, j=i
0<i<n &0<j<n?
N
Y
Form admittance matrix
i==j?
N
Y
Calculate according to (1) when RR[ ][0]==i||RR[ ][1]==j, during scanning RR
Calculate according to (2) when RR[ ][0]==i&RR[ ][1]==j, during scanning RR
i=i+1, j=j+1

Fig. 4 Calculation process for admittance matrix

values of voltage can be estimated. Otherwise, they can be set according to the correction equation.

$$\begin{cases} e_i^{(k)} = e_i^{(k-1)} + ee \\ f_i^{(k)} = f_i^{(k-1)} + ff \end{cases} \tag{4}$$

Step 4: Calculate the unbalance value of the PQ buses' active power (*PP*) and reactive power (*QQ*), and the PV buses' active power (*PP*) and voltage (*UU*), according to (5)-(7) as follows:

$$PP[i] = P[i] - \sum_{j=1}^{j=n} [e[i](YG[i][j]e[j] - YB[i][j]f[j]) + f[i](YG[i][j]f[j] - YB[i][j]e[j])] \tag{5}$$

$$QQ[i] = Q[i] - \sum_{j=1}^{j=n} [f[i](YG[i][j]e[j] - YB[i][j]f[j]) + e[i](YG[i][j]f[j] - YB[i][j]e[j])] \tag{6}$$

$$UU[i]^2 = U[i]^2 - (e[i]^2 + f[i]^2) \tag{7}$$

Figure 5 consists of inner and outer loops. The outer loop traverses the first subscript (indicating the bus) of the unbalance value and the inner loops compute the value of each element by accumulation. The first inner loop calculates the unbalance value of active power, then judges the type of bus. If it is a PQ bus, the second inner loop calculates the unbalance value of reactive power, otherwise the unbalance value of voltage.

Step 5: Calculate the elements of the Jacobian matrix using (8), (9) and save them in the related two-dimensional RDDs arrays $H[n+1][n+1]$, $N[n+1][n+1]$, $J[n+1][n+1]$, $L[n+1][n+1]$, $R[n+1][n+1]$ and $S[n+1][n+1]$. PQ buses are represented by elements *H*, *N*, *J*, *L* in Jacobian matrix and PV buses by *H*, *N*, *R*, *S*.

$$\begin{cases} H[i][j] = YG[i][j]f[i] - YB[i][j]e[i] \\ N[i][j] = YG[i][j]e[i] + YB[i][j]f[i] \\ J[i][j] = -N[i][j] \\ L[i][j] = H[i][j] \\ R[i][j] = 0\& \ S[i][j] = 0 \end{cases} \tag{8}$$

where $i \neq j$, when $i = j$, there exists:

$$\begin{cases} \dot{I} = Y[i][i]\dot{U}_i + \sum\limits_{\substack{j=1 \\ j\neq i}}^{j=n} Y[i][i]\dot{U}[j] = a[i][i] + jb[i][i] \\ H[i][i] = YG[i][i]f[i] - YB[i][i]e[i] + b[i][i] \\ N[i][i] = YG[i][i]e[i] + YB[i][i]f[i] + a[i][i] \\ J[i][i] = -YG[i][i]e[i] - YB[i][i]f[i] + a[i][i] \\ L[i][i] = YG[i][i]f[i] - YB[i][i]e[i] - b[i][i] \\ R[i][i] = 2f[i] \\ S[i][i] = 2e[i] \end{cases} \tag{9}$$

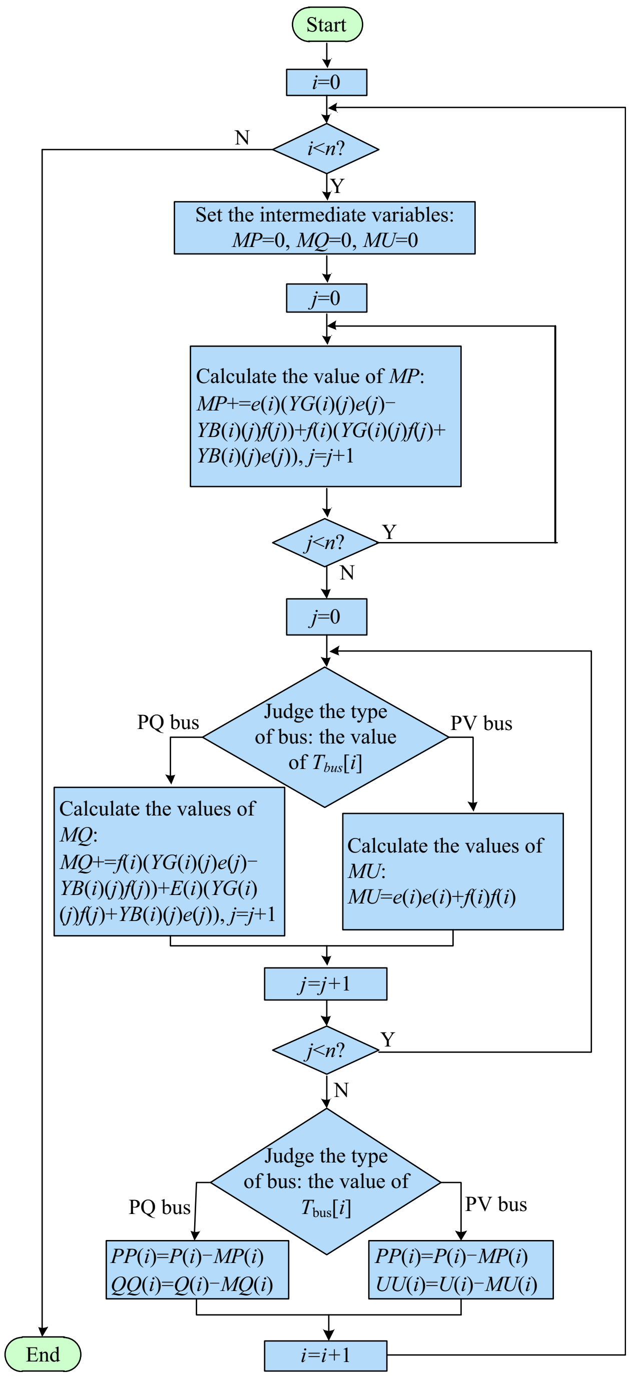
Start
i=0
i<n?
N
Y
Set the intermediate variables: MP=0, MQ=0, MU=0
j=0
Calculate the value of MP: MP+=e(i)(YG(i)(j)e(j)−YB(i)(j)f(j))+f(i)(YG(i)(j)f(j)+YB(i)(j)e(j)), j=j+1
j<n?
Y
N
j=0
Judge the type of bus: the value of T_bus[i]
PQ bus
PV bus
Calculate the values of MQ: MQ+=f(i)(YG(i)(j)e(j)−YB(i)(j)f(j))+E(i)(YG(i)(j)f(j)+YB(i)(j)e(j)), j=j+1
Calculate the values of MU: MU=e(i)e(i)+f(i)f(i)
j=j+1
j<n?
Y
N
Judge the type of bus: the value of T_bus[i]
PQ bus
PV bus
PP(i)=P(i)−MP(i)
QQ(i)=Q(i)−MQ(i)
PP(i)=P(i)−MP(i)
UU(i)=U(i)−MU(i)
End
i=i+1

Fig. 5 Calculation process of unbalance value

Step 6: Sort the Jacobian matrix elements from Step 5 according to the order of PQ and PV buses and the patterns $\begin{bmatrix} H, N \\ J, L \end{bmatrix}$ and $\begin{bmatrix} H, N \\ R, S \end{bmatrix}$ respectively to construct the Jacobian matrix RDDs array $Y[2n][2n]$. Similarly, construct the unbalance value vector RDDs array $PQU[2n][1]$ according to the order of PQ and PV buses and the patterns $\begin{bmatrix} PP \\ QQ \end{bmatrix}$ and $\begin{bmatrix} PP \\ UU \end{bmatrix}$ respectively. Following (2) multiply the inverse of the Jacobian matrix by PQU to obtain the RDDs array $ef[2n][2n]$ which are the correction values, arranged in order of PQ and PV buses and the pattern $\begin{bmatrix} ee \\ ff \end{bmatrix}$.

The process of calculation is shown in Fig. 6. There are two parts: generating the Jacobian matrix and getting its inverse. The generation of the Jacobian matrix consists of two loops: firstly traversing each line of the matrix and then calculating the elements. The process of inversion is to get each element's algebraic cofactor and form a quotient with the determinant of the Jacobian matrix. Finally, the inverse matrix is multiplied by the unbalanced value vector, thus solving the correction equation, and the results are saved in the RDDs array $ef[\,]$. The values of array $ef[\,]$ are placed into $ee[\,]$ and $ff[\,]$ according to the order of Y_{bus} values. For example, if $Y_{bus}[1] = 2$, then $ee[2] = ef[2], ff[2] = ef[3]$.

Referring to Fig. 3, the termination condition for the iteration compares the maximum correction with a specified threshold, and the iteration will finish and jump to Step 7 if this condition is met, otherwise, the iteration will continue from Step 3.

Step 7: To complete the power flow calculation, calculate the distribution of power and line loss in the power grid according to (10), (11) and (12).

$$\widetilde{S}_s = \dot{U}_s \sum_{i=1}^{n} \overset{*}{Y}_{si} \overset{*}{U}_i = P_s + \mathrm{j}Q_s \tag{10}$$

$$\widetilde{S}_{ij} = \dot{U}_i \overset{*}{I}_{ij} = \dot{U}_i \left[\overset{*}{U}_i \overset{*}{y}_{i0} + (\overset{*}{U}_i - \overset{*}{U}_j) \overset{*}{y}_{ij} \right] = P_{ij} + \mathrm{j}Q_{ij} \tag{11}$$

$$\widetilde{S}_{ji} = \dot{U}_j \overset{*}{I}_{ji} = \dot{U}_j \left[\overset{*}{U}_j \overset{*}{y}_{j0} + (\overset{*}{U}_j - \overset{*}{U}_i) \overset{*}{y}_{ji} \right] = P_{ji} + \mathrm{j}Q_{ji} \tag{12}$$

4 Experimental results and analysis

4.1 Cluster-based computing environment

The experimental platform is made up of a 5-node cluster in a cloud computing platform, which includes a master and four slaves. Figure 7 shows the topological graph.

Running on the cluster are four processes NameNodes and one ResourceManager, which is also called the Master of the cluster. There are also four processes DataNode, which act as Workers in the cluster, and one NodeManager. DataNode and NameNode are responsible for managing storage files. ResourceManager and NodeManager manage the cluster resources. The Master and Worker processes are started by Spark and then run the tasks submitted by Spark. The hardware configuration of each node includes: 20 GB disk, 1 processor and 1 GB memory. The cluster uses Hadoop and Spark technology, with hadoop distributed file system (HDFS) as its storage system for large-scale power data, Yarn as resource manager, Spark as memory-based distributed parallel computing framework and Scala as a light and convenient programming language. The details of software configuration are in Table 1.

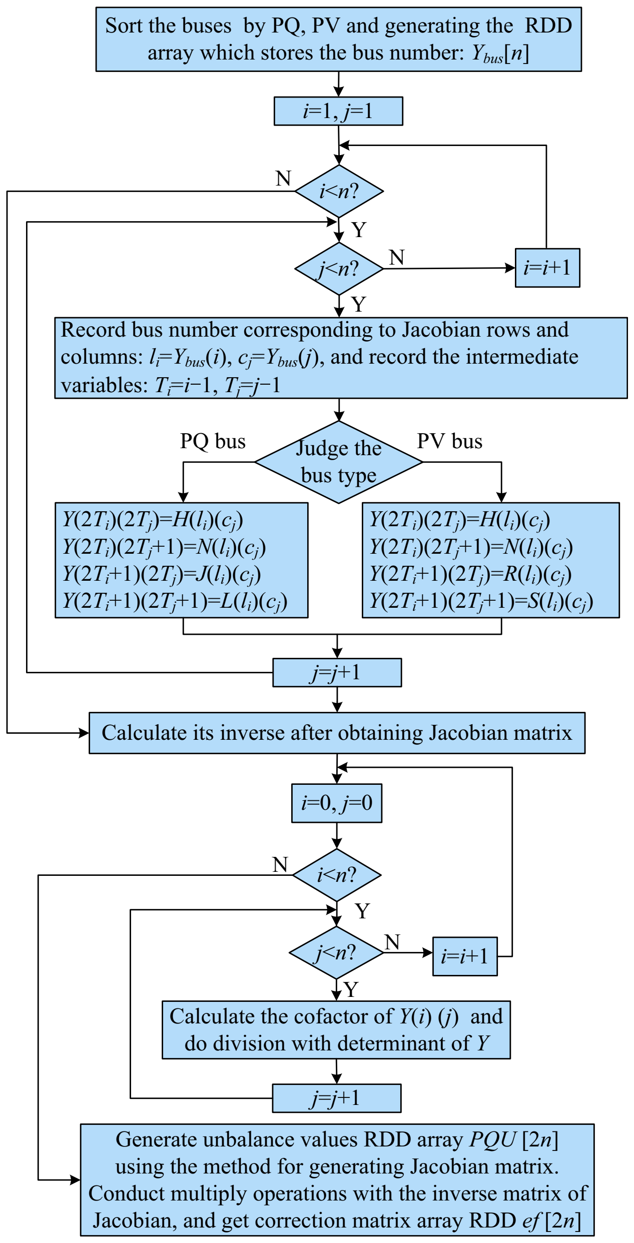
Sort the buses by PQ, PV and generating the RDD array which stores the bus number: $Y_{bus}[n]$
i=1, j=1
N
i<n?
Y
j<n?
N
i=i+1
Y
Record bus number corresponding to Jacobian rows and columns: l_i=$Y_{bus}(i)$, c_j=$Y_{bus}(j)$, and record the intermediate variables: T_i=i−1, T_j=j−1
PQ bus
Judge the bus type
PV bus
$Y(2T_i)(2T_j)=H(l_i)(c_j)$
$Y(2T_i)(2T_j+1)=N(l_i)(c_j)$
$Y(2T_i+1)(2T_j)=J(l_i)(c_j)$
$Y(2T_i+1)(2T_j+1)=L(l_i)(c_j)$
$Y(2T_i)(2T_j)=H(l_i)(c_j)$
$Y(2T_i)(2T_j+1)=N(l_i)(c_j)$
$Y(2T_i+1)(2T_j)=R(l_i)(c_j)$
$Y(2T_i+1)(2T_j+1)=S(l_i)(c_j)$
j=j+1
Calculate its inverse after obtaining Jacobian matrix
i=0, j=0
N
i<n?
Y
j<n?
N
i=i+1
Y
Calculate the cofactor of $Y(i)$ (j) and do division with determinant of Y
j=j+1
Generate unbalance values RDD array PQU [2n] using the method for generating Jacobian matrix. Conduct multiply operations with the inverse matrix of Jacobian, and get correction matrix array RDD ef [2n]

Fig. 6 Process of generating and inverting Jacobian matrix

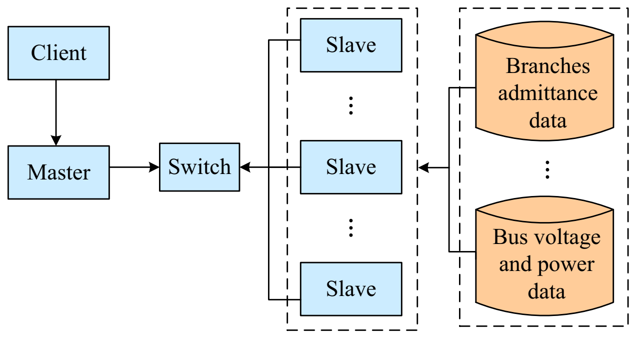

Fig. 7 Cluster nodes topology

Table 1 Software information of nodes

Software	Name	Version
Operating system	Centos	V6.0
Software development kit (SDK)	Jdk	1.8.0_31
Programming language	Scala	2.11.5
File system	HDFS	Hadoop-2.5.2
Resource manager	Yarn	Hadoop-2.5.2
Computing framework	Spark	1.2.0
Programming tool	IntelliJ IDEA	14.0.3

4.2 Source data and test cases

The test data are stored in two files: a branch impedance file and a bus voltage and power file. The columns stored in each file are listed in Tables 2 and 3. According to (1) and (2), the branch admittance value *YR* (real part) and *YB* (imaginary part) will be acquired and then the admittance matrix can be obtained. Data are accessed according to the type of bus and unused or non-existant data are set to 0 when reading the file.

In order to simulate large-scale power flow calculations, test cases are constructed from multiple instances of the standard test systems IEEE 30, IEEE 118 and IEEE 300. The test systems are interconnected by tie-lines to make $N \times N$ network structures, so that larger-scale case systems come into being. The number of buses and branches in each constructed test case are shown in Table 4.

4.3 Experimental processing and results analysis

To analyze the advantages of a distributed memory cluster, three aspects are taken into account: the scale of simulated power system, the speedup of the power flow calculation and the scale of Spark cluster. Power flow calculations using the method outlined above were completed on the Spark cluster and some run-time features of the experiment are shown in Tables 5 and 6.

Table 2 Arrangement of data in branch impedance file

Column	Meaning
1	From bus
2	To bus
3	Resistance
4	Reactance

Table 3 Arrangement of data in buses' voltage and power file

Column	Meaning
1	Bus type (1: PQ, 2: PV, 3: Balance bus)
2	Bus number
3	Voltage amplitude (p.u.)
4	Active power
5	Reactive power

Table 4 Information of cases

Case name	Number of buses	Number of branches
1062	1062	1620
2383	2383	2896
2737	2737	3506
3000	3000	4280
7680	7680	10976
161542	161542	247715
331462	331462	508323
498550	498550	764595

Table 5 shows information about available resources of each node in the Spark cluster after starting. The cluster has four slaves and they have been successfully started. Each node with "active" status can perform tasks assigned by the master at any moment. Table 6 shows details of five completed tasks. The Spark cluster allocates an ID for each task in its execution process. Each slave can use 1 GB memory. Each task is performed by all slaves at the same time using 4 kernels.

Testing the performance of parallel computing requires benchmark testing tools, such as SparkBench. However, SparkBench supports only four types of application: machine learning, graph calculation, SQL querying, and streaming data calculation. It can't be used to test power system flow calculations directly. SparkBench performance indices include the task execution time and resource consumption, yet not the volumes of shuffle data, input and output data, etc. From the above description, it is difficult to give an accurate and detail description of the I/O time

Table 5 Information about available nodes after starting Spark cluster successfully

Node ID	Status	Processor	Memory (GB)
Worker-20150711005200-202.206.212.11-50571	Active	1 (0 used)	1.0 (0.0 used)
Worker-20150711005825-202.206.212.8-33135	Active	1 (0 used)	1.0 (0.0 used)
Worker-20150711005825-202.206.212.10-49791	Active	1 (0 used)	1.0 (0.0 used)
Worker-20150711005825-202.206.212.7-53383	Active	1 (0 used)	1.0 (0.0 used)

Table 6 Information about finished tasks

Business ID	Name	Used memory of every node (GB)	Used processor	Status	Running time (s)
app-20150711021723-0004	OnLine1	1	4	Done	51
app-20150711015925-0003	OnLine2	1	4	Done	14
app-20150711012931-0002	OnLine3	1	4	Done	18
app-20150711012132-0001	OnLine4	1	4	Done	21
app-20150711010053-0000	OnLine5	1	4	Done	11

and the volume of intermediate data. Referencing [7, 9], this article adopts the commonly used running time and speedup ratio to evaluate the performance of parallel programs. Running time refers to the time that the whole parallel hardware platform takes to execute a parallel program and solve a problem. Specifically, it includes: the arithmetic calculation time, which is measured by the number of steps of basic operations during the execution of the algorithm; the communication time, which is measured by the count of communication traffic; and the I/O time, which is measured by the count of data read from or written to files and their transmission delay.

4.4 Experiment 1: comparison of running time for different models and cases

The foundation of this work and the problems which need to be solved are based on a cloud computing platform, so the proposed method and others are compared in the same cloud computing architecture and experimental environment, so that the validity and advantages of proposed method can be verified.

The first experiment compares the running time of the power flow calculation in stand-alone model, using the MapReduce model, and using the proposed method when the number of buses is 118, 1062, 2383, 2737, 3000, 7680, 161542, 331462, and 498550. For good convergence, the number of iterations of the correction equation is set to be 3. In Fig. 8, the abscissa is the number of buses and the ordinate is running time. It shows the change of running time with the number of buses under the three computational models.

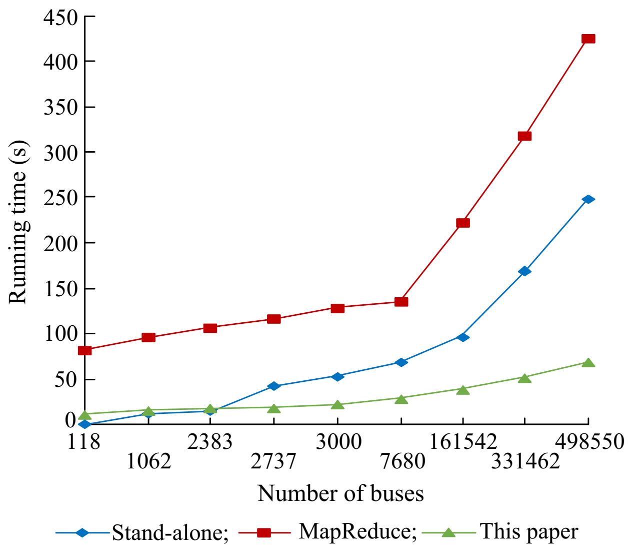

Fig. 8 Running time of power flow calculation for different numbers of buses using three computational models

It can be seen from Fig. 8 that, as the number of buses increases, the advantages of the proposed method emerge gradually and its performance is better than MapReduce and the stand-alone model when the number of buses is equal to or more than 2383. Meanwhile, it should be noted that when the amount of data is small the advantage of the parallel computing model with cloud computing are not obvious. Under the cluster model, after submitting the task, every node needs to establish connection with the cluster during the initialization process, at the same time, the HDFS must establish communication with Spark, and MapReduce and the cluster's internal nodes need to establish communication with each other, which altogether takes about 10 s. In the task execution process, the master will divide the task into subtasks and assign to slaves for execution. So, because of these overheads, it can be seen that when the number of buses is 118, the running time in stand-alone model is even less than for the other two models.

In all cases, the running time of the power flow calculation using the MapReduce model is higher than stand-alone model. When there is a small amount of data, the distribution achieved with the Map task is seriously uneven, so there is a great difference in the utilization ratio of nodes and the computing capability of cluster cannot be fully exploited, while a lot of management overhead still exists, resulting in low performance. For large-scale power flow calculation, MapReduce calculation tasks are executed in multiple nodes where data are stored, which will produce temporary data and frequent read/write operations for the HDFS. A large number of disk I/O operations are required and, as a result, its performance is still poor and has no advantage compared with the sequential stand-alone model.

Comparing with traditional parallel computation models such as MPI, the advantages of proposed method are also dependent on the data scale. A comparison of running time between methods from [7, 9, 12, 33] and the proposed method of this paper is shown in Table 7. It should be noted that the single value used [7, 9, 12, 33] in Table 7 is the lowest among them. Table 7 illustrates that when the number of buses is small to moderate (less than 106200), the performance of this method is slower than the best achieved in [7, 9, 12, 33]. When the maximum number of buses is 106200, [7, 9, 12, 33] achieve a lowest running time of 49.72 s, nevertheless, the proposed method can reach a lower running time of 38.376 s in the case of 161542 buses. As the number of buses continues to increase, the performance advantages of the proposed method can be better seen, and this is determined by the structure and inherent characteristics of the cloud computing environment.

Table 7 Running time of methods proposed in this paper and in [7, 9, 12, 33] when using four processors

Number of buses	Running time (s)	
	References [7, 9, 12, 33]	This paper
118	0.896	11.012
1062	1.330	14.842
3000	1.620	21.321
7680	8.310	27.912
106200	49.720	–
161542	–	38.376

4.5 Experiment 2: change of time speedup based on the proposed method

Speedup refers to the ratio between two times. This experiment measures the running time ratio of the proposed method and the stand-alone power flow calculation as the number of buses changes. The result is given in Fig. 9, where the abscissa is the number of buses and the ordinate is the time speedup of the power flow calculation.

The experimental results show that the speedup ratio increases with the number of buses, which indicates that the parallel performance of the proposed method is enhanced with an increasing amount of data and calculation. Presumably, if the number of buses continues to increase of buses, there is still room for the speedup to improve.

4.6 Experiment 3: efficiency of power flow calculation with different cluster scales

Due to limited experimental conditions, The cluster built with the Spark framework has only 4 slaves and 1 master, but in a practical environment the cluster would be far larger than this. As the scale of cluster expands, the advantages will become more obvious. This experiment compares the time efficiency of power flow calculation using the proposed method in the cases of 5 nodes (4 slave nodes and one master node) and 9 nodes (8 slave nodes and one master node). The test data come from the synthesized systems with 3000, 161542, and 331462 buses. Figure 10 shows the time needed when using the 5-node cluster and the 9-node cluster for these systems. As can be seen, the time needed for the power flow calculation reduces when the number of slave nodes increases from 4 to 8, but the reduction is not proportional to the number of added nodes. This is because additional time is needed for communicating with the new nodes.

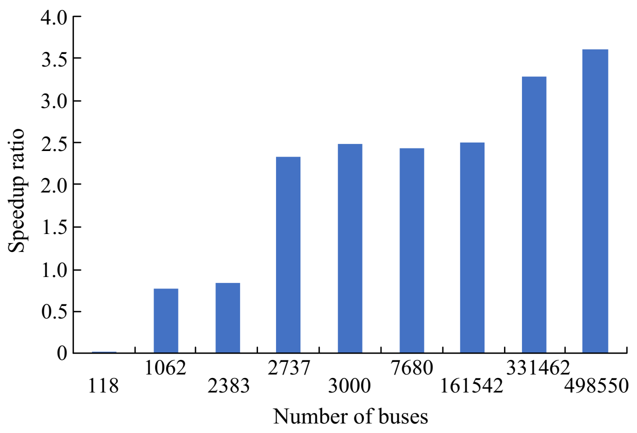

Fig. 9 Speedup ratio for proposed method compared to a stand-alone power flow calculation

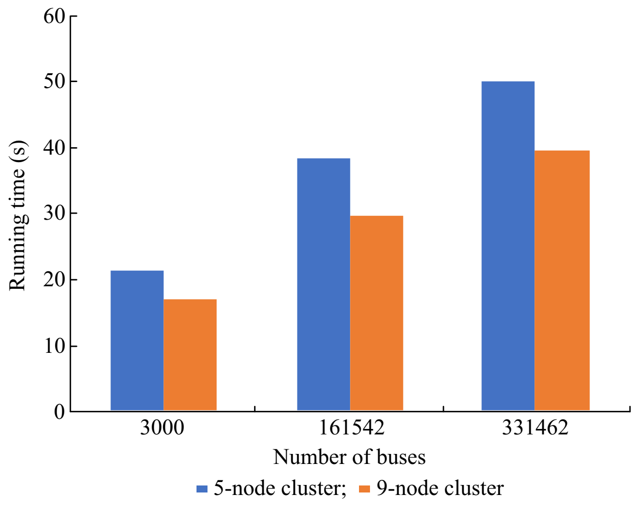

Fig. 10 Time needed for power flow calculation when using different cluster scales

4.7 Experiment summary

Through the above three experiments, we can draw the following conclusions:

1) The time performance of the power flow calculation using the proposed method is superior to the stand-alone model and the MapReduce model, and the speedup increases with the number of buses.

2) The time performance of power flow calculation using the proposed method is improved with the expansion of the Spark cluster. However, as the cluster enlarges, the cost for managing the cluster will also increase. The running time of the proposed method won't be reduced in proportion to the scale of cluster.

Additionally, some shortcomings about experimental process and results need to be pointed out and explained:

1) The power flow calculation has achieved better performance in stand-alone model when the number of buses is relatively small. This phenomenon confirms that cloud-based parallel computation, including the proposed method, is more appropriate for a large volume of data.

2) In this paper, we reference the experimental setups of power flow calculation in [7, 9, 12, 33], and compare with the best running time among these papers. Although the numbers of power flow calculation buses and branches are not exactly the same as in this paper, the running time of power flow calculations may be compared among similar configurations.

3) There is a big gap between the proposed method and mature commercial tools. This paper uses five cheap PCs to build an experimental cluster and deploys a virtual machine (CentOS operating system) on each node (Windows operating system) for the power flow calculation experiment. In the process of the power flow calculation, the virtual machine occupies only a small amount of CPU, memory (1 GB) and other resources available on the PCs. The overall performance of this cluster including its stability and reliability are only applicable for experimental testing, and its performance is not as good as some commercial models. Through employing a reliable and stable basic computing platform, and conducting practical improvement and optimization on it, the performance will be greatly improved. For example, a smart grid dispatching control system named D5000 developed by the NARI Group, and the solution for intelligent dispatching system in Central China Power Grid designed by Sugon corporation, have been employing traditional MPI and OpenMP respectively. Nevertheless, the performance of each system is greatly better than that reported in [7, 9, 12, 33].

In summary, in the same cloud computing experiment platform, the testing results show that this method has greatly improved the computational performance compared with MapReduce, and its running time is better than [7, 33] in the case of a large number of buses, which verifies the advantages of proposed method.

5 Conclusion

This paper proposed a parallel power flow calculation method based on RDDs, DAG optimization and the Newton-Raphson method to deal with the problem of large amounts of data transmission and low efficiency in the MapReduce model for parallel computation. Running time comparisons were performed in an experimental environment. There were some disparities between the simulation data and actual power grids, as well as between the experimental cluster configurations and the commercial tools for cluster computing. However, in similar experimental environment, the results show that the proposed method is more suitable for large-scale power systems compared with stand-alone, MapReduce and some methods proposed in the literature.

In the future, some studies about the allocation of virtual machines, task scheduling and load balancing strategy are hoped to be carried out to meet the needs of higher computing performance and analysis efficiency when applied to power flow calculation.

Acknowledgements This work was supported by National Natural Science Foundation of China (No. 51677072).

References

[1] Xu Y, Sun H, Liu H et al (2018) Distributed solution to DC optimal power flow with congestion management. Electr Power Energy Syst 95:73–82
[2] Abdi H, Beigvand SD, Scala ML (2017) A review of optimal power flow studies applied to smart grids and microgrids. Renew Sustain Energy Rev 71:742–766
[3] Shu J, Guan R, Wu L (2017) Optimal power flow in distribution network considering spatial electro-thermal coupling effect. IET Gener Transm Distrib 11(5):1162–1169
[4] Niu M, Wan C, Xu Z (2014) A review on applications of heuristic optimization algorithms for optimal power flow in modern power systems. J Mod Power Syst Clean Energy 2(4):289–297
[5] Sun Y, Tang X (2014) Cascading failure analysis of power flow on wind power based on complex network theory. J Mod Power Syst Clean Energy 2(4):411–421
[6] Sun Q, Chen H, Yang J (2014) Analysis on convergence of Newton-like power flow algorithm. Proc CSEE 34(13):2196–2200
[7] Ao L, Cheng B, Li F (2010) Research of power flow parallel computing based on MPI and P-Q decomposition method. In: Proceedings of 2010 international conference on electrical and control engineering, Washington, USA, 25–27 June 2010, 4 pp
[8] Shayesteh E, Gayme DF, Amelin M (2018) System reduction techniques for storage allocation in large power systems. Electr Power Energy Syst 95:108–117
[9] Xie K, Zhang H, Hu B et al (2010) Distributed algorithm for power flow of large-scale power systems using the GESP technique. Trans China Electrotech Soc 25(6):89–95
[10] Wang S, Guo J, Song D et al (2014) The research of power system simulation application under the peer-to-peer computing. China Electric Power 47(5):48–52
[11] Xia P, Wang F (2012) Large-scale power system rapid flow calculation method research. Power Syst Prot Control 40(9):38–42
[12] Hu B, Xie K, Cao K (2011) Parallel GMRES techniques for solving Newton power flow of large scale power systems on the Beowulf cluster. Trans China Electrotech Soc 26(4):145–152
[13] Sanseverino ER, Buono L, Silvestre MLD et al (2017) A distributed minimum losses optimal power flow for islanded microgrids. Electr Power Syst Res 152:271–283
[14] Roberge V, Tarbouchi M, Okou F (2016) Optimal power flow based on parallel metaheuristics for graphics processing units. Electr Power Syst Res 140:344–353
[15] Abdelaziz M (2017) GPU-OpenCL accelerated probabilistic power flow analysis using Monte-Carlo simulation. Electr Power Syst Res 147:70–72
[16] Zhou K, Yang S (2015) A framework of service-oriented operation model of China's power system. Renew Sustain Energy Rev 50:719–725
[17] Qu Z, Zhu L, Zhang S (2013) Data processing of Hadoop-based wide area measurement system. Autom Electr Power Syst 37(4):92–97
[18] Mei H, Mi Z, Wu G (2014) Massive data processing of intermittent energy based on MapReduce model. Autom Electr Power Syst 38(15):76–80
[19] Ousterhout K, Rasti R, Ratnasamy S et al (2015) Making sense of performance in data analytics frameworks. In: Proceedings of 12th USENIX conference on networked systems design and implementation, Berkeley, USA, 4–6 May 2015, 15 pp
[20] Zaharia M, Chowdhury M, Das T et al (2012) Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster. In: Proceedings of 9th USENIX conference on networked systems design and implementation, Berkeley, USA, 25–27 April 2012, 2 pp
[21] Ananthanarayanan G, Ghodsi A, Borthakur D et al (2012) PACMan: coordinated memory caching for parallel jobs. In: Proceedings of 9th USENIX conference on networked systems design and implementation, Berkeley, USA, 24–27 April 2012, 1 pp
[22] Wei L, Lian W, Liu K et al (2014) Hippo: an enhancement of pipeline-aware in-memory caching for HDFS. In: Proceedings of 23rd international conference on computer communication and networks, Shanghai, China, 4–7 August 2014, 5 pp
[23] Tang Z (2014) Design and implementation of machine learning platform based on Spark. Dissertation, Xiamen University
[24] Feng L (2013) Research and implementation of memory optimization based on parallel computing engine Spark. Dissertation, Tsinghua University
[25] Lou Z (2009) Research on power flow with electric power transmission & distribution networks feature. Dissertation, Xi'an University of Science and Technology
[26] Phuc T, Thang V, Binh P et al (2015) A computing tool for composite power system reliability evaluation based on Monte Carlo simulation and parallel processing. Lecture Notes in Electrical Engineering, Springer, Cham
[27] Blaskiewicz P, Zawada M, Balcerek P et al (2015) An application of GPU parallel computing to power flow calculation in HVDC networks. In: Proceedings of 2015 23rd euromicro international conference on parallel, distributed and network-based processing, Turku, Finland, 4–6 March 2015, 7 pp
[28] Meng X, Tang W, Liu Y et al (2015) Parallel computing of three-phase unbalanced power flow in large-scale complex distribution network. Power Syst Prot Control 43(13):45–51
[29] Wu J, Xia M, Xu L et al (2014) Power system analysis. Tsinghua University Press, Beijing
[30] Zhang Y (2013) Research of power flow calculation for distribution grids with distributed generations. Dissertation, South China University of Technology
[31] Yan L, Xie M, Xu J (2013) Improved power flow calculation of distribution network with DG. Power Syst Prot Control 41(5):17–22
[32] Xu T (2012) The power flow analysis based on cloud computing. Dissertation, Hefei University of Technology
[33] Liu Y, Zhou J, Xie K et al (2006) Parallel PCG solution of large scale power system equations based on Beowulf cluster. Trans China Electrotech Soc 21(3):105–111

Dewen WANG received his Ph.D. degree in Power System and Automation from North China Electric Power University, Baoding, China, in 2009. He is now an associate professor in North China Electric Power University. His research interests include power system automation and intelligent information processing.

Fangfang ZHOU received her Master degree in North China Electric Power University, Baoding, China, in 2018. Her research interests include big data analysis technology.

Jiangman LI received her B.E. degree in University of South China, Hengyang, China, in 2013. And she also has received her Master degree in North China Electric Power University, Baoding, China, in 2016. Her research interests include cloud computing technology.