Resilient Distributed Datasets (RDD)

Appear as variable in you program but are actually stored in multiple machine

myrdd = sc.textFile("Word and Peace.txt")

RDD { immutable, parallel transformations, rebuilt on failure, persistance

Transformations: map, filter, groupBy, join

Actions: Count, Collect, Save

Turn local collection to RDD

num = sc.parallelize([1, 2, 3])

square = num.map(lambda x: x*x) => 1, 4, 9

even = square.filter(lambda x: x%2==0) => 4

num.collect() 1, 2, 3

num.take(2) 1, 2

num.count() 3

num.reduce(lambda x, y: x+y) 6

num.saveAsTextFile("123.txt")

myrdd = sc.textFile("word and Peace.txt")

rdd is a immutable list Transform

can't change it, can't see individual elements, can only interact with specific ops

newrdd = myrdd.map(f) [1, 2, 3] -> [f(1), f(2), f(3)]

Return a new rdd by applying a function to each element of the rdd

newrdd = myrdd.map(eval)

['1,4', '2,3', '1,3'] => [1,4, 2,3, 1,3]

把 string 变数字 array[] -> array[1,0]

newrdd = myrdd.flatMap(lambda x: x.split())

Map each element to multiple elements.

['a b c d'] -> ['a', 'b', 'c', 'd']

distinct()

remove duplicates

Sample(with Replacement, fraction) 有无放回之后 分数

抽出数量由 fraction 和数据(随机)

sortBy(key-func, ascending = true)

Sort RDD w.r.t keyfunc

sortBy(lambda x: -x) 逆序

How does map work

1. Serialize to binary representation

2. Ship to workers

3. De-serialize

4. Apply to every elements

Function is sent to data, data don't move.

Variables defined in the driver program will automatically be shipped to the cluster along with function definition.

- Each task get a new copy (updates are not sent back)
- Variable must be pick-able
- 〃 can't be rdd
- Don't use field of a object
- Beware of shipping large variables.

Val = myrdd.reduce → Action

Reduce using commutative (交换), associative (结合) operations return to drive program

How does reduce work:

Step 1: move result to $n/2$ processors, combine result

Step 2: Repeat

n processors $k = \log_2 n$

Total message: $\frac{n}{2} + \frac{n}{4} + \cdots + 1 = n - 1$

rdd.takeOrdered(3) 1, 1, 2 从小到大 3个

takeOrdered(3, key = lambda x: -x)

6, 5, 3 从大到小 3个

reduce vs aggregate

reduce: input, output of aggregations are of the same type

aggregations allows them to be different

aggregate(zeroValue, seqOp, comOp)

Starting value OT OT, IT OT OT

Actions for numerical RDD only

mean(), sum(), max(), min(), variance(), sampleVariance(), stdev(), sampleStdev()

key value pairs

[(a, 1), (b, 1), (a, 2)]

reduceByKey(lambda x, y: x+y)

[(a, 3), (b, 1)]

It's a transform not an action

lines.flatMap(lambda x: x.split()) \

.map(lambda x: (x, 1)) \

.reduceByKey(lambda x, y: x+y)

How does reduceByKey work

- Each key is mapped to a machine (partition, actually) using python builtin hash function
- A shuffle take place: key value pairs are moved appropriate machines, collocating pairs with identity keys
- Reduce then applied locally at each machine.

Hash: - value are first combined locally before the shuffle takes place

- shuffle avoid when not needed (partition awareness)

reduceByKey()

groupByKey() [(a, seq(1, 2)), (b, seq(1))]

sortByKey() [(a, 1), (a, 2), (b, 1)]

mapValues(lam x: x+1) [(a, 2), (a, 3), (b, 2)]

flatMapValues(lam x: range(x+1)) [(a, 0), (a, 1), (b, 0), (b, 1), ...]

values (1, 1, 2),

keys (a, b, a), collectAsMap()

('a': 1, 'b': 1)

allFiles = sc.wholeTextFiles('dir')

combineByKey(createCombiner, mergeValue, mergeCombiner).

join

a.join(b)

a: A 2, B 4, B 6 ⋈ b: B UG, C G => B(4, UG), B(6, UG)

How does join work

Join requires shuffling, shuff avoid when not necessary through partition awareness

① (C, G), (A, 2) (B, 4), (B, 6) ② (B, UG) => ③ (B(4, UG)), (B(6, UG))

rdd = sc.parallelize([(1, 2), (3, 4), (3, 6)])

other = (3, 9), (5, 8)

rdd.join(other) (3, (4, 9)) (3, (6, 9))

rdd.leftOuterJoin(other) (1, (2, None)) (3, (4, 9)) (3, (6, 9))

rdd.rightOuterJoin (3, (4, 9)), (3, (6, 9)), (5, (None, 8))

rdd.subtractByKey(other) - [(1, 2)]

rdd.cogroup(other) (1, ([2], [])), (3, ([4, 6], [9])), (5, ([], [8]))

reduceByKey(lam x, y: x+y, 5) can control level of parallelism

groupByKey(5)

RDD are internally split into partitions

#partition < #machine

Part 0 Part 1

Part 0 Part 1 2 3 4 5

Map serially in each partition.

If machine store k partition and n>k processors partition evaluation executed in parallel.

m workers with k processors

ideal partition = m k

Fewer partition: not exploiting full parallelism in this operation

More 〃: No advantage in speedup, maybe advantage in memory usage

partitionBy(#partitions, partitionFunction = Hash)

partitionBy(5) 5 partition with default hash

partitionBy(5, partitionFunc = lam x: hash(x)+10)

When rdd is shuffled by partitionBy, there are stored in private variables. when reduceByKey is called next it check to see if these fields are set. if so it skips shuffle

Join of rdd with the same partitioning information do not require shuffle (Useful doing repeat join on large rdd)

Create & preserve: cogroup(), groupWith(), join(), leftOuterJoin(), rightOuterJoin(), groupByKey(), reduceByKey(), combineByKey(), partitionByKey(), sort()

Preserve: mapValues(), flatMapValues(), filter.

non key value operations: 1. map will remove partition info even if it does not alter keys

Lazy evaluation

- Transform evaluation until action (reduce, collect, count) requires their computation.
- Instead Spark builds a DAG (direct acyclic graph of rdd dependencies

sc. →parallelize .map →rdd with tallies → value output return to driver variables:

rdd.map: 惰性

Pipelining

Non-pipeline executions worker can be idle

Pipelined execution: A worker can start OP before other worker finish

RDD are not stored But DAG is

Pipelining: Execution postponed can be grouped together and parallelized more efficiency

Resilience (R in RDD): DAG allows recovery from crashed nodes.

persistence

RDD that are reused can be stored in memory by explicitly calling cache()

做完map之后cache 之后的 action 不用重复计算

Run out of money

If cache is full, rdds are evicted. using LRU policy

想访问 evicted RDD

1、Resilience! DAG used to recompute

2、Option to spill excess RDD in hard disk persist

Page Ranking

initial $S_w = \frac{1}{N}$

$d_4 = 3$

Main Loop for $r = 0.15$

$S_w = r \cdot \frac{1}{N} + (1-r) \sum_{w': w' \to w} \frac{S_{w'}}{d_{w'}}$

指向w的点　w指向的点数

小结：① S_w高如果很多点连向 w

② 如果 S_w高的点指向 w，w的 S_w高

Back ground

$\langle x, y \rangle = x^T y = \sum x_i y_i$

$(AB)^T = B^T A^T$

Linear function: ① $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$

② $f(x) = Ax$　f linear

③ $f(x) = b^T x$, f linear

Affine function: $f(x) = b^T x + C$ / $Ax + C$.

linear function + constant

Quadratic function: $f(x) = \frac{1}{2} x^T Q x + b^T x + c$

symmetric　vector

$f: \mathbb{R}^n \to \mathbb{R}$ is called norm if

$f(x) \geq 0$ for all x　f is non-negative

$f(x) = 0$ implies $x = 0$　f is definite

$f(tx) = |t| f(x)$　f is homogeneous 齐次

$f(x+y) \leq f(x) + f(y)$

$\|x\|_2 = \sqrt{x^T x} = \sqrt{\sum x_i^2}$

$\|x\|_1 = \sum_{i=1}^{n} |x_i| + \cdots + |x_n|$

$\|x\|_\infty = \max\{\|x_1\|, \ldots \|x_n\|\}$

$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p}$

$\alpha \|x\|_a \leq \|x\|_b \leq \beta \|x\|_a$

converge

$\lim_{k \to \infty} x_k = x$　　$x_k \to x$

continuous

$\lim_{k \to \infty} x_k = x \Rightarrow \lim_{k \to \infty} f(x_k) = f(x)$

Gradient

$\frac{\partial f(x)}{\partial x_i} = \lim_{\delta \to 0} \frac{f(x + \delta e_i) - f(x)}{\delta}$

$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$

$f(x) = b^T x + c$　$\nabla f(x) = b$

$f(x) = \frac{1}{2} x^T Q x + b^T x + c$　$\nabla f(x) = Qx + b$

taylor expansion

$f(x) = f(x_0) + (\nabla f(x))^T (x - x_0) + o(\|x - x_0\|_2)$

Hessian $\nabla^2 f(x) = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$

$AA^{-1} = A^{-1}A = I$

$Ae = \lambda e$　$\|e\| = 1$

对称矩阵 $A = Q \Lambda Q^T$, $Q^T Q = I$

$\Lambda = \mathrm{diag}(\lambda_1 \ldots \lambda_n)$　$\lambda_1 > \cdots > \lambda_n$

$Q = [e_1 \ldots e_n]$ eigenvector

$\det(A) = \prod \lambda_i$　$\mathrm{trace}(A) = \sum \lambda_i$

$x^T A x \geq 0$ PSD
$x^T A x > 0$ PD　> A 对称

---

line segment $x = \theta x_1 + (1-\theta) x_2$　$0 < \theta < 1$

Convex set 包含 X中所有点

✓　✗

convex combination $x = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

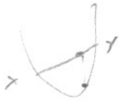
convex hull conv S: set of all convex combination s of point in S

convex cone　$x = \theta_1 x_1 + \theta_2 x_2$

---

hyperplane $a^T x = b$

halfspace $a^T x \leq b$



convex function:

$f(\theta x + (1-\theta) y) \leq \theta f(x) + (1-\theta) f(y)$ ①

Strictly convex　〃　<　〃

1st order condition

$f(y) \geq f(x) + \nabla f(x)^T (y - x)$

2nd order

$\nabla^2 f(x) \geq 0$　f convex

> 0　strictly convex

$\|Ax - b\|_2^2$　$\nabla f(x) = 2A^T(Ax - b)$

$\nabla^2 f(x) = 2A^T A$

Jensen's 不等式

f convex. 有 ①

$f(Ez) \leq E f(z)$

$f(x) = h(g(x))$

f convex $\begin{cases} g \text{ convex}, h \text{ convex}, h \text{ nondecreasing} \\ g \text{ concave}, h \text{ convex}, h \text{ nonincrease} \end{cases}$

$f''(x) = h''(g(x)) g'(x)^2 + h'(g(x)) g''(x)$

---

gradient descent

方向 $-\nabla f(x)$

$x^{k+1} = x^k + \alpha^k d^k$

$d^k = -\alpha^k \nabla f(x^k)$　方向 $-\nabla f(x^k)$

步长

Decreasing step size $\alpha^k = \frac{1}{k+1}$

Exact line search $t_{opt} = \arg\min_{t > 0} f(x + t\Delta x)$

Back-tracking line search

starting $t = 1$, repeat $t = \beta t$ until

$f(x + t\Delta x) < f(x) + \alpha t \nabla f(x) \Delta x$　t 步长 $\Delta x = -\nabla f(x)$

$\alpha \in (0, 1/2)$　$\beta \in (0, 1)$

until $\|\nabla f(x)\|_2 < \epsilon$

---

strongly convex　$\nabla^2 f(x) \geq mI$.　$m > 0$

$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2} \|x - y\|_2^2$

$f(x^k) - f(p^*) \leq c^k (f(x^0) - f(p^*))$　$c \in (0, 1)$

$g(x) = f(x) + \lambda \|x\|_2^2$, $\lambda > 0$, f convex $\Rightarrow$ g strongly convex

---

~~牛顿~~ newton method

$\Delta x_{nt} = -\nabla^2 f(x)^{-1} \nabla f(x)$