ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

VŨ THỊ THANH

MÔ HÌNH NGÔN NGỮ SỬ DỤNG MAPREDUCE

Ngành: Công nghệ thông tin
Chuyên ngành: Kỹ thuật phần mềm
Mã Số: 60480103

LUẬN VĂN THẠC SĨ

NGƯỜI HƯỚNG DẪN KHOA HỌC CHÍNH: **TS. NGUYỄN VĂN VINH**
NGƯỜI HƯỚNG DẪN KHOA HỌC PHỤ: **TS. NGUYỄN PHÚ BÌNH**

Hà Nội – 2016

MỤC LỤC

MỤC LỤC ..... **i**
LỜI CẢM ƠN ..... **iii**
LỜI CAM ĐOAN ..... **iv**
DANH MỤC THUẬT NGỮ VIẾT TẮT ..... **v**
DANH MỤC HÌNH VẼ ..... **vi**
DANH MỤC BẢNG ..... **vii**
GIỚI THIỆU **8**
***Chương 1:*Mô hình ngôn ngữ** **10**
1.1 Giới thiệu: 10
1.2 Mô hình ngôn ngữ N-gram 11
1.3 Khó khăn khi xây dựng mô hình ngôn ngữ N-gram 13
1.3.1 Phân bố không đều: 13
1.3.2 Kích thước bộ nhớ của mô hình ngôn ngữ 13
1.4 Các phương pháp làm mịn 14
1.4.1 Phương pháp Add-one 14
1.4.2 Phương pháp Good – Turing 15
1.4.3 Phương pháp truy hồi back-off 16
1.4.4 Phương pháp nội suy 18
1.4.5 Phương pháp Kneser – Ney 19
1.4.6 Phương pháp Kneser – Ney cải tiến 20
1.5 Đánh giá mô hình ngôn ngữ 21
1.5.1 Entropy – Độ đo thông tin: 21
1.5.2 Perplexity – Độ hỗn loạn thông tin: 22
1.5.3 Error rate – Tỉ lệ lỗi: 23
***Chương 2:*Tổng quan về Hadoop MapReduce** **24**
2.1 Hadoop 24
2.2 Các thành phần của Hadoop 24
2.2.1 Kiến trúc hệ thống tệp phân tán 24

2.3 Mapreduce 26
2.3.1 Kiến trúc của Mapreduce 27
2.3.2 Cơ chế hoạt động 28
2.4 Ưu điểm của Hadoop 31
Chương 3: **Ước lượng mô hình ngôn ngữ với Mapreduce 32**
3.1 Đếm các từ 33
3.2 Đếm số lần xuất hiện (Generate count of counts) 36
3.3 Sinh số làm mịn Good-Turing 37
3.4 Ước lượng xác suất n-gram 38
3.5 Sinh bảng Hbase 40
3.5.1 Cấu trúc dựa trên n-gram 40
3.5.2 Cấu trúc dựa trên từ hiện tại 40
3.5.3 Cấu trúc dựa trên đoạn văn 41
3.5.4 Cấu trúc dựa trên nửa ngram 42
3.5.5 Cấu trúc dựa trên số nguyên 43
3.6 Truy vấn trực tiếp 44
Chương 4: **Các phương pháp đánh giá và thực nghiệm 46**
4.1 Các phương pháp đánh giá 46
4.1.1 Thời gian và bộ nhớ 46
4.1.2 Sự so sánh độ hỗn loạn thông tin mô hình ngôn ngữ 46
4.2 Thực nghiệm 47
4.2.1 Môi trường chạy thực nghiệm 47
4.2.2 Dữ liệu 47
4.2.3 Đánh giá thời gian và bộ nhớ cho các ngram 48
4.2.4 So sánh thời gian chạy với SRILM 50
KẾT LUẬN 52
TÀI LIỆU THAM KHẢO 53

LỜI CẢM ƠN

Đầu tiên, cho phép tôi gửi lời cảm ơn sâu sắc tới TS Nguyễn Văn Vinh và TS Nguyễn Phú Bình, người đã trực tiếp hướng dẫn, chỉ bảo và tạo điều kiện cho tôi trong quá trình hoàn thành luận văn này.

Đồng thời tôi cũng xin gửi lời cảm ơn chân thành tới các thầy cô giáo trường Đại học Công Nghệ, Đai học Quốc Gia Hà Nội, những người đã trực tiếp giảng dạy, hướng dẫn và tạo điều kiện cho tôi trong quá trình học tập và làm luận văn.

Cuối cùng, tôi xin gửi lời cảm ơn tới tất cả các bạn đồng học và gia đình đã ủng hộ, giúp đỡ tôi hoàn thành luận văn.

LỜI CAM ĐOAN

Tôi xin cam đoan kết quả trong luận văn là sản phẩm của riêng cá nhân tôi. Trong toàn bộ nội dung của luận văn, những điều được trình bày hoặc là của cá nhân hoặc là được tổng hợp từ nhiều nguồn tài liệu. Tất cả các tài liệu tham khảo đều có xuất xứ rõ ràng và được trích dẫn hợp pháp.

Tôi xin hoàn toàn chịu trách nhiệm theo quy định cho lời cam đoan của mình.

Hà Nội, ngày 25 tháng 10 năm 2016
Người cam đoan

Vũ Thị Thanh

DANH MỤC THUẬT NGỮ VIẾT TẮT

STT	Từ viết tắt	Từ đầy đủ	Ý nghĩa
1	WER	Word Error Rate.	Tỉ lệ lỗi
2	ASR	Automatic Speech Recognition	Nhận dạng tiếng nói tự động
3	MLE	Maximum Likelihood Estimation	Ước lượng hợp lý hóa cực đại.
4	MSE	Mean Squared Error	Sai số toàn phương trung bình
5	HDFS	Hadoop Distributed File System	Hệ thống tệp phân tán Hadoop
6	FIFO	First in first out	Vào trước ra trước

DANH MỤC HÌNH VẼ

Hình 2.1: Kiến trúc Hadoop 24
Hình 2.2: Kiến trúc của HDFS 25
Hình 2.3: Mô hình Mapreduce 26
Hình 2.4: Kiến trúc MapReduce.....27
Hình 2.5: Cơ chế hoạt động của MapReduce.....28
Hình 2.6: Mối quan hệ giữa JobTracker và Task Tracker.....29
Hình 2.7: Mô hình Task Tracker 29
Hình 2.8: Mô hình JobTracker 30
Hình 2.9: Cơ chế hoạt động của JobTracker 31
Hình 5.1: Số lượng n-gram.....48
Hình 5.2: Thời gian chạy đếm thô.....49
Hình 5.3: Không gian lưu trữ cho đếm thô. 50

DANH MỤC BẢNG

Bảng 4.1: Cấu trúc bảng dựa trên n-gram. 40

Bảng 4.2: Cấu trúc bảng dựa trên từ.....41

Bảng 4.3: Cấu trúc bảng dựa trên đoạn văn 42

Bảng 4.4: Cấu trúc bảng dựa trên nửa ngram.....43

Bảng 4.5: Cấu trúc bảng dựa trên số nguyên.....44

Bảng 5.1: Dữ liệu.....47

Bảng 5. 2: Số lượng n-gram cho các thứ tự khác nhau 48

Bảng 5.3: Thời gian chạy đếm thô 49

Bảng 5.4: không gian lưu trữ trong đếm thô 50

GIỚI THIỆU

Ngày nay với sự phát triển của công nghệ thông tin, lượng dữ liệu trao đổi trên mạng là rất lớn. Dữ liệu về văn bản, hình ảnh, âm thanh đang trở thành những nguồn dữ liệu khổng lồ để phục vụ các nhu cầu về lưu trữ và trao đổi thông tin của con người. Đã có nhiều ứng dụng ra đời để hỗ trợ con người các công việc như: kiểm tra chính tả trên các văn bản, nhận dạng dữ liệu, nhận dạng giọng nói, dịch máy thống kê. Để phát triển các ứng dụng đó người ta đã đưa ra mô hình ngôn ngữ như là một tiền đề để ứng dụng vào các lĩnh vực trên. Mô hình ngôn ngữ là một vấn đề quan trọng của lĩnh vực xử lý ngôn ngữ tự nhiên. Mô hình ngôn ngữ là các phân bố xác suất của một đoạn văn trên một tập văn bản lớn. Vì vậy một mô hình ngôn ngữ tốt sẽ đánh giá câu đúng ngữ pháp và độ trôi chảy tốt hơn những câu có thứ tự ngẫu nhiên. Cách thông dụng nhất được dùng để mô hình hóa ngôn ngữ là thông qua các N-gram. Mô hình N- gram sử dụng các tập dữ liệu văn bản lớn để ước lượng xác suất của mô hình. Nhìn chung thì dữ liệu càng lớn thì mô hình sẽ càng tốt hơn [13].

Khi xây dựng mô hình ngôn ngữ cần phải có một lượng bộ nhớ khá lớn để có thể lưu trữ được xác suất của tất cả các chuỗi và cần cấu hình máy phải mạnh để tính toán và xử lý. Có nhiều phương pháp, kỹ thuật đã được đưa ra để tối ưu bộ nhớ và bộ xử lý. Các phương pháp làm mịn, truy hồi, đồng hóa, nén là những phương pháp trước đây dùng để tối ưu giá trị xác suất và tối ưu bit lưu trữ. Một số ứng dụng về xây dựng mô hình ngôn ngữ được sử dụng gần đây như công cụ SRILM, Random Forest Language Model Toolkit, … Mục đích chính của SRILM là để hỗ trợ ước lượng và đánh giá mô hình ngôn ngữ. Random Forest Language Model Toolkit xây dựng dựa trên công cụ SRILM,là một trong các mô hình ngôn ngữ cây quyết định cho kết quả thực nghiệm khá tốt. Tuy nhiên hạn chế của các công cụ trên là với dữ liệu rất lớn thì sẽ tốn rất nhiều thời gian để thực hiện. Với những dữ liệu cực lớn thì có thể sẽ không chạy được. Để giải quyết bài toán với dữ liệu huấn luyện lớn thì hadoop và mapreduce là một công cụ tối ưu nhất. Đó chính là lý do tại sao tôi lựa chọn đề tài " Mô hình ngôn ngữ sử dụng MapReduce" cho nghiên cứu của mình.

Đề tài này nhằm mục đích nghiên cứu sử dụng Hadoop và MapReduce vào việc xây dựng mô hình ngôn ngữ nhằm cải tiến tốc độ cho việc xây dựng mô hình ngôn ngữ và ước lượng mô hình để có thể thực hiện với lượng dữ liệu rất lớn để đưa ra mô hình ngôn ngữ chính xác hơn. Trong phần ứng dụng xây dựng mô hình ngôn ngữ với MapReduce luận văn sẽ sử dụng phương pháp làm mịn GoodTuring. Có nhiều phương pháp làm mịn có thể cho kết quả tốt hơn như Kneser-Ney nhưng do thời gian có hạn nên luận văn đã sử dụng phương pháp làm mịn GoodTuring để đơn giản cho việc xây dựng chương trình nhưng cũng đủ tốt để xây dựng mô hình ngôn ngữ.

Nội dung luận văn này được trình bày trong bốn chương. Phần giới thiệu về đề tài. Phần này trình bày các ngữ cảnh, các nghiên cứu đã có về vấn đề cần giải quyết, lý do lựa chọn đề tài, mục tiêu của đề tài và cấu trúc nội dung của luận văn.

Chương 1 trình bày các khái niệm cơ bản phục vụ cho đề tài. Chương này sẽ trình bày các kiến thức cơ bản về mô hình ngôn ngữ, mô hình N-gram, các phương pháp làm mịn và các độ đo dùng để đánh giá mô hình ngôn ngữ.

Chương 2 trình bày các kiến thức cơ bản về Hadoop và MapReduce, giới thiệu về kiến trúc của Hadoop, MapReduce cũng như cơ chế làm việc của chúng.

Chương 3 sẽ trình bày về việc ứng dụng Hadoop và MapReduce vào mô hình ngôn ngữ.

Chương 4giới thiệuvề công cụ thực nghiệm và kết quả thực nghiệm.

Phần kết luậnđưa rakết luận, định hướng phát triển cho đề tài. Cuối cùng là tài liệu tham khảo.

Chương 1: Mô hình ngôn ngữ

Trong xử lý ngôn ngữ tự nhiên, mô hình ngôn ngữ được sử dụng rộng rãi. Mô hình ngôn ngữ được áp dụng trong nhiều lĩnh vực như nhận dạng giọng nói, dịch máy. Mô hình ngôn ngữ ngày càng được nhận được nhiều sự quan tâm bởi các nhà khoa học. Trong chương này tôi sẽ trình bày về các kiến thức cơ bản về mô hình ngôn ngữ như định nghĩa mô hình ngôn ngữ, mô hình n-gram, các phương pháp đánh giá mô hình ngôn ngữ và các phương pháp làm mịn.

1.1 Giới thiệu:

Mô hình ngôn ngữ là một phân bố xác suất của một đoạn văn bản trên một tập dữ liệu văn bản lớn. Ví dụ, trong ngôn ngữ tiếng việt thì xác suất của câu "Tôi ăn cơm" sẽ cao hơn câu "cơm ăn tôi".

Thuật ngữ mô hình ngôn ngữ bắt nguồn từ các mô hình xác suất sinh ngôn ngữ dùng cho hệ thống nhận dạng tiếng nói được phát triển vào những năm 1980. Vào đầu thế kỷ 20 Andrey Markov đưa ra mô hình Markov sử dụng để lập mô hình cho chuỗi các chữ cái. Sau đó Claude Shannon đưa ra mô hình cho các chữ cái và các từ.

Mô hình ngôn ngữ được định nghĩa như sau: Tập V là tập các từ trong ngôn ngữ Ví dụ khi xây dựng một mô hình ngôn ngữ cho tiếng anh chúng ta có thể có

V = {the, dog, laughs, saw, barks, cat…}

Tập V có thể rất lớn: nó có thể chứa vài nghìn hoặc chục nghìn từ và là tập hữu hạn. Một câu trong ngôn ngữ là một tập các từ đứng gần nhau $w_1w_2...w_n$ (với $n \geq 1$ và $w_i \in V$ với i = {1,…,(n-1)}), một ký hiệu <s> ở đầu câu và </s> ở cuối câu (hai ký hiệu <s> và </s> không thuộc tập V).

Ví dụ:

<s> the dog barks </s>

<s> the cat laughs </s>

<s> the cat saw the dog </s>

Tập V^+ là tập các câu sinh ra từ các từ trong tập V, đây là tập vô hạn bởi vì các câu có thể có độ dài bất kỳ.

Từ đó chúng ta có định nghĩa sau:

Mô hình ngôn ngữ: Là mô hình gồm một tập hữu hạn V và một hàm $P(w_1w_2…w_n)$ như sau:

1. Với cụm $(w_1w_2…w_n) \in V^+$, $P(w_1w_2…w_n) \geq 0$
2. $\sum_{w1w2…wn \in V+} P(w1w2 \dots wn) = 1$

Khi đó $P(w_1w_2…w_n)$ là một phân bố xác suất của câu trên tập V^+.

Gọi $C(w_1w_2…w_n)$ là số lần xuất hiện của câu $w_1w_2…w_n$ trong tập huấn luyện, N là tổng các câu. Mô hình ngôn ngữ trên tập dữ liệu huấn luyện có thể định nghĩa như sau:

$$\mathbf{P(w_1w_2…w_n) = \frac{C(w_1w_2…w_n)}{N}} \quad \mathbf{(1.1)}$$

Tuy nhiên đây không phải là một mô hình tốt vì trên thực tế nó sẽ cho xác suất bằng 0 cho các câu không có trong tập huấn luyện, do đó không thể tổng quát hóa cho trường hợp câu không có trong tập V^+. Mặc dù có hạn chế nhưng mô hình ngôn ngữ vẫn được xem xét để nghiên cứu cải tiến vì những lý do sau:

1. Mô hình ngôn ngữ rất cần thiết cho những ứng dụng lớn như nhận diện giọng nói và dịch máy.
2. Từ các kỹ thuật định nghĩa hàm P và cho sự ước lượng các tham số từ tập huấn luyện sẽ cho kết quả với nhiều ngữ cảnh khác nhau như mô hình ngôn ngữ Markov ẩn.

1.2 Mô hình ngôn ngữ N-gram

Nhiệm vụ của mô hình ngôn ngữ là cho biết xác suất của một câu $w_1w_{2…}w_n$ là bao nhiêu. Theo công thức Bayes: P(AB) = P(B|A)*P(A), thì có thể suy ra được

$$\mathbf{P(w_1w_2…w_m) = P(w_1)*P(w_2|w_1)* P(w_3|w_1\ w_2) * ….* P(w_m|w_1w_{2…}w_{m-1}).} \quad \mathbf{(1.2)}$$

Theo công thức này thì bài toán tính xác suất của mỗi chuỗi từ quy về bài toán tính xác suất của một từ với điều kiện biết các từ trước nó. Do đó mô hình cần một lượng bộ nhớ khá lớn để lưu xác xuất của tất cả các cụm từ. Rõ ràng công thức này vẫn không hiệu quả khi chiều dài của các cụm từ lớn. Để có thể tính được xác suất của văn bản với bộ nhớ chấp nhận được thì ta có thể sử dụng xấp xỉ Markov với giả định rằng xác suất của một từ chỉ phụ thuộc vào hữu hạn từ trước đó chứ không phải phụ thuộc toàn bộ vào dãy đứng trước. Xấp xỉ Markov có thể dự đoán xác suất của một từ khi biết 1,…,n từ trước đó. Như vậy công thức tính xác suất văn bản (1.2) tương đương với công thức sau:

$$\mathbf{P(w_1w_2…w_m) = P(w_1) * P(w_2|w_1) * P(w_3|w_1w_2) *…* P(w_{m-1}|w_{m-n-1}w_{m-n}…w_{m-2})* P(w_m|w_{m-n}w_{m-n+1}…w_{m-1})} \quad \mathbf{(1.3)}$$

Mô hình Markov còn được gọi là mô hình N-gram [1][2].

Ví dụ cho câu sau: “This is a sentence”, mô hình N-gram cho câu đó như sau

N = 1(unigrams): This,
is,
a,
sentence

N = 2 (bigrams): This is,
is a,
a sentence
N = 3 (trigrams): This is a,
is a sentence

Áp dụng công thức xấp xỉ Markov cho các mô hình N-gram sẽ tương đương với các công thức sau:

Với N = 1: Mô hình ngôn ngữ Unigram

$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k)$

$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1)\, P(w_2) \ldots P(w_T)$

Với N = 2: Mô hình ngôn ngữ Bigram

$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-1})$

$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1|<S>)\, P(w_2|w_1) \ldots P(w_T|w_{T-1})$

Với N = 3: Mô hình ngôn ngữ Trigram

$P(w_k|w_1,\ldots,w_{k-1}) \approx P(w_k|w_{k-2}, w_{k-1})$

$\Rightarrow P(w_1,w_2,\ldots,w_T) \approx P(w_1|<S><S>) \ldots P(w_T| w_{T-2}w_{T-1})$

Xây dựng mô hình N-Gram

Sử dụng những câu có sẵn để tính các ước lượng xác xuất n-gram
Chúng ta sử dụng các thuật ngữ sau [1]:

- N = Tổng số các từ trong tập huấn luyện
- V = Tập từ vựng
- $C(w_1,\ldots,w_k)$ = số lần xuất hiện của n-gram $w_1,\ldots,w_k$ trong tập huấn luyện
- $P(w_1,\ldots,w_k)$ = ước lượng xác suất cho n-gram $w_1,\ldots,w_k$
- $P(w_k| w_1,\ldots,w_{k-1})$ = xác xuất của w_k với lịch sử $w_1,\ldots,w_{k-1}$

Áp dụng ước lượng hợp lý hóa cực đại cho xác xuất n-gram cụ thể như sau:

- Unigram: $P(w_i) = \frac{C(w_i)}{N}$
- Bigram: $P(w_i,w_j) = \frac{C(w_i,w_j)}{N}$

$$P(w_j|w_i) = \frac{P(w_i, w_j)}{P(w_i)} = \frac{C(w_i, w_j)}{C(w_i)}$$

Sử dụng tần số tương đối khi ước lượng

Ước lượng hợp lý hóa cực đại của tập dữ liệu huấn luyện cho mô hình là P(D|M)

Xét ví dụ với tập huấn luyện như sau:

<s> I am sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

Xác xuất 2-gram của tập dữ liệu trên sẽ là

$P(I|<s>) = \frac{2}{3} = 0.67$ $P(Sam|<s>) = \frac{1}{3} = 0.33$

$P(am|I) = \frac{2}{3} = 0.67$ $P(do|I) = \frac{1}{3} = 0.33$

$P(</s>|Sam) = \frac{1}{2} = 0.5$ $P(sam|am) = \frac{1}{2} = 0.5$

1.3Khó khăn khi xây dựng mô hình ngôn ngữ N-gram

1.3.1 Phân bố không đều:

Khi sử dụng mô hình N-gram sự phân bố không đều trong tập văn bản huấn luyện có thể dẫn đến các ước lượng không chính xác. Khi các N-gram phân bố thưa, nhiều cụm n-gram không xuất hiện hoặc chỉ có số lần xuất hiện nhỏ, việc ước lượng các câu có chứa các cụm n-gram này sẽ có kết quả tồi. Với V là kích thước bộ từ vựng, ta sẽ có V^n cụm N-gram có thể sinh từ bộ từ vựng. Tuy nhiên, thực tế thì số cụm N-gram có nghĩa và thường gặp chỉ chiếm rất ít.

Ví dụ: tiếng Việt có khoảng hơn 5000 âm tiết khác nhau, ta có tổng số cụm 3-gram có thể có là: $5.000^3 = 125.000.000.000$ Tuy nhiên, số cụm 3-gram thống kê được chỉ xấp xỉ 1.500.000. Như vậy sẽ có rất nhiều cụm 3-gram không xuất hiện hoặc chỉ xuất hiện rất ít.

Khi tính toán xác suất của một câu, có rất nhiều trường hợp sẽ gặp cụm Ngram chưa xuất hiện trong dữ liệu huấn luyện bao giờ. Điều này làm xác suất của cả câu bằng 0, trong khi câu đó có thể là một câu hoàn toàn đúng về mặt ngữ pháp và ngữ nghĩa. Đề khắc phục tình trạng này, người ta phải sử dụng một số phương pháp "làm mịn"

1.3.2Kích thước bộ nhớ của mô hình ngôn ngữ

Khi kích thước tập văn bản huấn luyện lớn, số lượng các cụm Ngram và kích thước của mô hình ngôn ngữ cũng rất lớn. Nó không những gây khó khăn trong việc lưu trữ mà còn làm tốc độ xử lý của mô hình ngôn ngữ giảm xuống do bộ nhớ của máy

tính là hạn chế. Để xây dựng mô hình ngôn ngữ hiệu quả, chúng ta phải giảm kích thước của mô hình ngôn ngữ mà vẫn đảm bảo độ chính xác.

1.4 Các phương pháp làm mịn

Để khắc phục tình trạng các cụm N-gram phân bố thưa như đã đề cập ở phần 1.3.1 người ta đã đưa ra các phương pháp làm mịn**.** Thuật ngữ làm mịn (smoothing) sử dụng cho việc đánh giá lại xác suất của các cụm N-gram. Các phương pháp làm mịn có thể chia thành các loại như sau:

Chiết khấu (Discounting): giảm xác suất của các cụm N-gram có xác suất lớn hơn 0 để bù cho các cụm N-gram không xuất hiện trong tập huấn luyện. Ví dụ: phương pháp Add-one, Good-Turing.

Truy hồi (Back-off): tính toán xác suất của các cụm N-gram không xuất hiện trong tập huấn luyện dựa vào các cụm N-gram ngắn hơn có xác suất lớn hơn 0. Ví dụ: Katz back-off.

Nội suy (Interpolation): tính toán xác suất của tất cả các cụm N-gram dựa vào xác suất của các cụm N-gram ngắn hơn.

1.4.1 Phương pháp Add-one

Phương pháp làm mịn Add-one hay còn gọi là phương pháp làm mịn Laplace Smoothing thực hiện cộng thêm 1 vào tần số xuất hiện của tất cả các cụm N-gram [3][4]

Xác suất của các cụm 1 từ w_i với tần suất xuất hiện là c_i là:

$$P(w_i) = \frac{C_i}{N}$$

Phương pháp Add-one thêm 1 vào các c_i, với V là số các từ trong bộ dữ liệu từ điển, ta có xác suất như sau:

$$\mathbf{P_{Add\text{-}one}(w_i) = \frac{C_i+1}{N+V}}$$

Đặt $C^* = (C_i+1)\frac{N}{N+V}$

Thì khi đó công thức xác xuất sẽ là

$$\mathbf{P^*(w_i) = \frac{C^*}{N}}$$

Với các cụm 2-gram thì ta có công thức sau

$$\mathbf{P(w_i,w_j) = \frac{C(w_i,w_j)}{N} \Rightarrow P_{Add\text{-}one}(w_i, w_j) = \frac{C(w_i, w_j)+1}{N+V^2}}$$

Khi đó $\mathbf{P_{Add\text{-}one}(w_j|w_i) = \frac{P_{Add\text{-}one}(w_i, w_j)}{P_{Add\text{-}one}(w_i)} = \frac{C(w_i,w_j)+1}{C(w_i)+V}}$

Xét các cụm N-gram với N>1thì xác suất của cụm $w_{i-n+1}...w_{i-1}w_i$ được tính theo công thức sau:

$$\mathbf{P(w_i|w_{i-n+1}...w_{i-1}) = \frac{C(w_{i-n+1}...w_{i-1}w_i) + 1}{C(w_{i-n+1}...w_{i-1}) + V}} \quad \mathbf{(1.4)}$$

Chúng ta có thể thấy thuật toán này sẽ làm thay đổi đáng kể xác suất của các cụm N-gram đã xuất hiện trong tập huấn luyện nếu kích thước bộ từ điển V là rất lớn. Trong thực nghiệm, một vài cụm N-gram có xác suất giảm đi gần 10 lần, do kích thước bộ từ điển là lớn trong khi tần số xuất hiện của cụm Ngram đó không cao. Để thuật toán thêm hiệu quả, người ta sử dụng công thức sau:

$$\mathbf{P(w_1w_2...w_n) = \frac{C(w_1w_2...w_n) + \lambda}{C(w_1w_2...w_{n-1}) + M\lambda}} \quad \mathbf{(1.5)}$$

Công thức trên là một phiên bản cải tiến thông dụng của thuật toán add-one. Để bảo toàn tổng xác suất của tất cả các cụm N-gram, thì λ được chọn trong khoảng [0, 1], với một số giá trị thông dụng sau:

- λ = 0: không làm mịn
- λ = 1: thuật toán add-one
- $\lambda = \frac{1}{2}$: được gọi là thuật toán Jeffreys – Perks

Phương pháp Add-one có ưu điểm là dễ cài đặt tính toán. Nhược điểm là làm giảm xác suất của những cụm từ hay xuất hiện trong tập huấn luyện. Nếu tỉ lệ các từ không xuất hiện càng lớn thì xác suất gán cho các từ này sẽ tăng và làm giảm đáng kể xác suất của các từ khác.

1.4.2Phương pháp Good – Turing

Ý tưởng của các phương pháp làm mịn bằng phương pháp chiết khấu là đếm tần suất xuất hiện của các từ có trong tập huấn luyện để tính xác suất của các từ chưa xuất hiện. Thuật toán Good-Turing [5][6] được đưa ra đầu tiên bởi Good. Thuật toán Good-Turing thực hiện ước lượng lại xác suất của những cụm từ (N-gram) có tần suất bằng 0 dựa trên số các từ có tần suất xuất hiện bằng 1.

Thuật toán Good-Turing dựa trên việc tính toán Nc, với Nc là số cụm N-gram xuất hiện c lần. Như vậy:

N_0 là số cụm n-gram có tần số 0 (số cụm N-gram không xuất hiện lần nào)

N_1 là số cụm n-gram có tần số 1 (số cụm N-gram xuất hiện 1 lần)

Tổng quát ta có :

$$Nc = \sum_{x:count\ (x)=c} 1$$

Khi đó, với mỗi c một ước lượng tần sốđược tính như sau:

$$\mathbf{c^* = (c+1)\frac{N_{c+1}}{N_c}}$$

Dùng công thức trên thay thế công thức MLE với bigram thì ta có công thức xác xuất sau:

$$\mathbf{P_{GT}(w_i, w_j) = \frac{C_{GT}(w_i,w_j)}{N}}$$

$$\mathbf{P_{GT}(w_j|w_i) = \frac{C_{GT}(w_i,w_j)}{C(w_i)}}$$

Với những bigram chưa xuất hiện:

$$c^* = C_{GT} = (0+1)\frac{N_1}{N_0}$$

$$P_{GT} = \frac{C_{GT}}{N}$$

Số cụm N-gram không xuất hiện lần nào trong bigram được tính như sau

$N_0 = V^2$ – những bigram đã xuất hiện

Trên thực tế, người ta không tính toán và thay thế mọi tần số c bởi một tần số mới c*. Người ta chọn một ngưỡng k nhất định, và chỉ thay thế tần số c bởi tần số mới c* khi c nhỏ hơn hoặc bằng k, còn nếu c lớn hơn k thì giữ nguyên tần số. Để đơn giản, người ta chọn k đủ lớn dựa vào kết quả huấn luyện.

1.4.3 Phương pháp truy hồi back-off

Giống như thuật toán chiết khấu, thuật toán truy hồi được sử dụng để giải quyết các vấn đề của tần suất bằng 0 trong N-gram. Ý tưởng của thuật toán backoff là tìm một (N-1) – gram nếu không có N- gram trong một chuỗi. Tiếp tục lùi lại các N-gram trước đó cho đến khi có tần suất lớn hơn 0.

Ví dụ với trigram chúng ta không có chuỗi $w_{n-2}w_{n-1}w_n$ để tính $P(w_n|w_{n-2}w_{n-1})$ thì có thể dùng xác suất bigram $P(w_n|w_{n-1})$. Tương tự như vậy nếu không thể tính $P(w_n|w_{n-1})$ chúng ta có thể dùng unigram $P(w_n)$.

Thuật toán backoff được đưa ra bởi Katz và công thức tính xác suất được đưa ra như sau:

$$\mathbf{P_{katz}(w_n|w_{n-N-1}) = \begin{cases} P^*(w_n|w_{n-N-1}^{n-1}) & \text{nếu } C(w^n_{n-N+1}) > 0 \\ \alpha(w^{n-1}_{n-N+1}) P_{katz}(w_n|w^{n-1}_{n-N+2}) & \text{nếu } C(w^n_{n-N+1}) = 0 \end{cases}} \quad \mathbf{(1.6)}$$

Áp dụng mô hình này cho 3-gram. Với "x, y, z" là một 3-gram thì

$$P_{katz}(z|xy) = \begin{cases} P^*(z|xy) & \text{nếu } C(xyz) > 0 \\ \alpha(x,y)P_{katz}(z|y) & \text{nếu } C(xyz) = 0 \text{ và } C(xy) > 0 \\ P^*(z) & \text{trường hợp còn lại} \end{cases}$$

Với 2 – gram thì:

$$P_{katz}(z|y) = \begin{cases} P_{GT}(z|y) & \text{nếu } C(yz) > 0 \\ \alpha(y)P_{GT}(z) & \text{nếu ngược lại} \end{cases}$$

Katz kết hợp phương pháp chiết khấu và giá trị α để cho tổng xác suất bằng 1. Vì nếu sử dụng xác suất MLE và dùng truy hồi về các gram nhỏ hơn thì xác suất sẽ được tính thêm một lượng, do đó tổng xác suất sẽ khác 1. Hệ số α sẽ đảm bảo tổng xác suất ở mức dưới bằng lượng để chiết khấu cho mức trên.

Sự chính xác của mô hình phụ thuộc vào hệ số α. Có một số kỹ thuật để chọn α tùy theo tập huấn luyện và mô hình ngôn ngữ. Một cách đơn giản là chọn α là một hằng số. Tuy nhiên rất khó để chọn một hằng số sao cho tổng xác suất của tất cả các N-gram không đổi. Gọi β là hàm biểu diễn tổng xác suất bên trái của hàm xác suất khối, β là một hàm của cụm (N-1) –gram. Hàm β tính bằng 1 trừ đi tổng xác suất khối giảm tại mức N –gram.

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n : c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+1}^{n-1})$$

Mỗi cụm từ trong (N-1) – gram sẽ nhận một phần nhỏ trong khối xác suất. Do đó ta có α như sau:

$$\alpha(w_{n-N+1}^{n-1}) = \frac{\beta(w_{n-N+1}^{n-1})}{\sum_{w_n:c(w_{n-N+1}^{n})=0} P_{katz}(w_n|w_{n-N+2}^{n-1})}$$

$$= \frac{1 - \sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+1}^{n-1})}{1 - \sum_{w_n:c(w_{n-N+1}^{n})>0} P^*(w_n|w_{n-N+2}^{n-1})}$$

1.4.4 Phương pháp nội suy

Các phương pháp chiết khấu được để cập trong mục trên giúp giải quyết được vấn đề của các cụm từ có tần suất xuất hiện bằng 0. Giả sử phải tính xác suất có điều kiện $P(w_n| w_{n-1}w_{n-2})$ nhưng không có cụm từ $w_{n-2}w_{n-1}w_n$ trong tập huấn luyện. Xác xuất này có thể tính thông qua xác suất của $P(w_n|w_{n-1})$. Nếu không tính được xác suất $P(w_n|w_{n-1})$ ta sử dụng $P(w_n)$. Có hai cách để thực hiện điều này là dùng phương pháp truy hồi và phương pháp nội suy. Phương pháp truy hồi sẽ thực hiện truy hồi xuống mức thấp khi mà tần suất của cụm từ đó bằng 0. Ngược lại phương pháp nội suy thực hiện kết hợp các xác xuất ở các N-gram.

Công thức tính xác suất theo phương pháp nội suy như sau:

$$\mathbf{P_I(w_i|w_{i-n+1}...w_{i-1}) = \lambda P(w_i|w_{i-n+1}...w_{i-1}) + (1-\lambda)P_I(w_i|w_{i-n+2}...w_{i-1})}$$

Áp dụng cho bigram và trigram ta có:

$$\mathbf{P_I(w_i|w_{i-1}) = \lambda P(w_i|w_{i-1}) + (1-\lambda)P(w_i)}$$

$$\mathbf{P_I(w_i|w_{i-n+1}...w_{i-1}) = \lambda_1 P(w_i|w_{i-2}w_{i-1}) + \lambda_2 P(w_i|w_{i-1}) + \lambda_3 P(w_i) \text{ với } \sum_i \lambda_i = 1}$$

Ở công thức trên, do tổng của tất cả các tham số λ bằng 1 nên để đơn giản ta có thể chọn tất cả λ bằng nhau và bằng $\frac{1}{3}$.

Tuy nhiên, cũng có thể chọn các tham số λ như là một hàm của Ngram:

$\lambda_1 = \lambda_1(w_{i-2}w_{i-1}w_i)$, $\lambda_2 = \lambda_2(w_{i-1}w_i)$ và $\lambda_3 = \lambda_3(w_i)$

1.4.5 Phương pháp Kneser – Ney

Kneser và Ney đưa ra phương pháp nội suy bằng các kết hợp xác suất ở gram mức dưới và xác suất ở gram mức trên [7][8]. Ví dụ khi xây dựng mô hình 2-gram trên tập dữ liệu huấn luyện xem xét trường hợp từ Francisco luôn xuất hiện sau từ San. Khi c(Francisco) cao thì xác suất 1-gram P(Francisco) cũng sẽ cao. Tuy nhiên trong trường hợp c(Francisco) thấp và từ Francisco chỉ đứng sau mỗi từ San nhưng xác suất 2-gram thì lại cao. Phương pháp Kneser-Ney xác suất của từ không tính dựa trên tần suất xuất hiện của từ đó mà dựa trên số từ khác nhau mà nó đứng liền kề sau. Phương pháp này được xây dựng theo hai mô hình là truy hồi và nội suy.

- Mô hình truy hồi:

$$\mathbf{P_{BKN}(w_i|w_{i-n+1}..w_{i-1}) = \begin{cases} \dfrac{C(w_{i-n+1}...w_i) - D}{C(w_{i-n+1}...w_{i-1})} & \text{nếu } C(w_{i-n+1}...w_i) > 0 \\ \alpha(w_{i-n+1}...w_{i-1}) P_{BKN}(w_i|w_{i-n+2}...w_{i-1}) & \text{nếu } C(w_{i-n+1}...w_i) = 0 \end{cases}} \quad \mathbf{(1.7)}$$

Trong đó:

  - $P_{BKN}(w_i) = \dfrac{N(vw_i) - D}{\sum_w N(vw)}$ với N(vw) là số lượng từ v khác nhau xuất hiện trước w trong tập huấn luyện

Như vậy:

$$P_{BKN}(w_i|w_{i-2}w_{i-1}) = \begin{cases} \dfrac{C(w_{i-2}w_{i-1}w_i) - D}{C(w_{i-2}w_{i-1})} & \text{nếu } C(w_{i-2}w_{i-1}w_i) > 0 \\ \alpha(w_{i-2}w_{i-1}) P_{BKN}(w_i|w_{i-1}) & \text{nếu } C(w_{i-2}w_{i-1}w_i) = 0 \end{cases}$$

$$P_{BKN}(w_i|w_{i-1}) = \begin{cases} \dfrac{C(w_{i-1}w_i) - D}{C(w_{i-1})} & \text{nếu } C(w_{i-1}w_i) > 0 \\ \alpha(w_{i-1}) P_{BKN}(w_i) & \text{nếu } C(w_{i-1}w_i) = 0 \end{cases}$$

$$P_{BKN}(w_i) = \frac{N(vw_i) - D}{\sum_w N(vw)}$$

- Mô hình nội suy:

$$\mathbf{P_{IKN}(w_i|w_{i-n+1}..w_{i-1}) = \frac{C(w_{i-n+1}..w_i) - D}{C(w_{i-n+1}..w_{i-1})} + \lambda(w_{i-n+1}..w_{i-1}) P_{IKN}(w_i|w_{i-n+2}..w_{i-1})} \quad \mathbf{(1.8)}$$

Trong đó:

  - $\lambda(w_{i-n+1}..w_{i-1}) = \dfrac{D\, N(w_{i-n+1}..w_{i-1}v)}{C(w_{i-n+1}..w_{i-1})}$ với $N(w_{i-n+1}..w_{i-1}v)$ là số lượng từ v khác nhau xuất hiện liền sau cụm $w_{i-n+1}..w_i$ trong tập huấn luyện

- $P_{IKN}(w_i) = \frac{N(vw_i) - D}{\sum_w N(vw)} + \lambda\frac{1}{V}$ với N(vw) là số lượng từ v khác nhau xuất hiện liền trước từ w trong tập huấn luyện.
- $\lambda = \frac{D\ N(v)}{\sum_w N(vw)}$

Như vậy:

$$P_{IKN}(w_i|w_{i-2}w_{i-1}) = \frac{C(w_{i-2}w_{i-1}w_i) - D}{C(w_{i-2}w_{i-1})} + \lambda(w_{i-2}w_{i-1})P_{IKN}(w_i|w_{i-1})$$

$$P_{IKN}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - D}{C(w_{i-1})} + \lambda(w_{i-1})P_{IKN}(w_i)$$

$$P_{IKN}(w_i) = \frac{N(vw_i) - D}{\sum_w N(vw)} + \lambda\frac{1}{V}$$

Trong cả 2 mô hình nội suy và truy hồi, D được chọn: $D = \frac{N_1}{N1 + 2N_2}$

1.4.6 Phương pháp Kneser – Ney cải tiến

Phương pháp làm mịn Kneser-Ney cải tiến được Chen và Goodman đưa ra năm 1999. Phương pháp này được pháp triển từ thuật toán Kneser-Ney. Thay vì sử dụng chiết khấu đơn D cho tất cả các từ có số lần xuất hiện bằng 0 trong phương pháp Kneser-Ney, phương pháp này đưa ra ba giá trị chiết khấu D_1, D_2, D_3 cho các N-gram có số lần xuất hiện bằng 1, 2 và 3.

Chen và GoodMan chọn D như sau:

$$D = \begin{cases} 0 \text{ nếu } c(w_{i-n+1}..w_i) = 0 \\ D_1 \text{ nếu } c(w_{i-n+1}.. w_i) = 1 \\ D_2 \text{ nếu } c(w_{i-n+1}.. w_i) = 2 \\ D_3 \text{ nếu } c(w_{i-n+1}.. w_i) >= 3 \end{cases}$$

Với $Y = \frac{N_1}{(N_1 + 2N_2)}$

$$D_1 = 1 - 2Y\frac{N_2}{N_1}$$

$$D_2 = 1 - 3Y\frac{N_3}{N_2}$$

$$D_3 = 1 - 4Y\frac{N_4}{N_3}$$

Trong đó: Ni là số lượng cụm N-gram có số lần xuất hiện

1.5 Đánh giá mô hình ngôn ngữ

Rất nhiều mô hình ngôn ngữ đã được đưa ra thì một câu hỏi cho những người sử dụng là làm sao để biết được mô hình nào tốt hay dở. Cách tốt nhất là đưa mô hình đó nhúng vào một ứng dụng khác để đánh giá. Ví dụ với hệ thống nhận dạng tiếng nói người ta thực hiện so sánh hiệu năng của hai mô hình ngôn ngữ bằng cách chạy lần lượt từng mô hình và xem kết quả trả về. Hạn chế của cách đánh giá này là phải nhờ đến hệ thống bên ngoài và thường chi phí đắt và khá lâu. Vì vậy các nhà nghiên cứu đã đưa ra các phương pháp đánh giá hiệu quả của mô hình ngôn ngữ độc lập với ứng dụng. Các phương pháp đó là

- **Entropy** - Độ đo thông tin
- **Perplexity** - Độ hỗn loạn thông tin
- **Error rate** - Tỉ lệ lỗi

1.5.1 Entropy – Độ đo thông tin:

Entropy là thước đo thông tin, có giá trị rất lớn trong xử lý ngôn ngữ. Nó thể hiện mức độ thông tin trong ngữ pháp, thể hiện sự phù hợp của một câu với một ngôn ngữ, và dự đoán được từ tiếp theo trong cụm Ngram[1]. Entropy của một biến ngẫu nhiên X được tính theo công thức**:**

$$H(X) = - \sum_{x \in X} p(x)\log_2 p(x)$$

Xét các câu gồm hữu hạn m từ $\mathbf{W = (w_1, w_2,..., w_m)}$ trong ngôn ngữ L. Ta có công thức tính entropy như sau:

$$H(w_1, w_2,..., w_m) = - \sum_{W \in L} p(w_1, w_2, ..., w_m)\log_2 p(w_1, w_2, ..., w_m)$$

Từ công thức trên, ta có thể đưa ra công thức tính tỉ lệ entropy trên các từ như sau:

$$\frac{1}{m} H(w_1, w_2,...,w_m) = - \frac{1}{m} \sum p(w_1, w_2, ..., w_m)\log_2 p(w_1, w_2, ..., w_m)$$

Thực tế thì tỉ lệ entropy trên các từ thường được sử dụng vì giá trị của nó không phụ thuộc vào độ dài các câu. Tuy nhiên, để tính được entropy của một ngôn ngữ L theo công thức trên thì ta phải xét tới các câu dài vô hạn (tất cả các câu có thể có trong ngôn ngữ L), đó là điều không thể. Do đó, ta có thể tính xấp xỉ tỉ lệ entropy trên các từ theo công thức sau:

$$H(L) = - \lim_{m \to \infty} \frac{1}{m} H(w_1, w_2, ..., w_m)$$

$$= - \lim_{m \to \infty} \frac{1}{m} \sum_{W \in L} p(w_1, w_2, ..., w_m) \log_2 p(w_1, w_2, ..., w_m)$$

Định lý **Shannon-McMillan-Breiman** đã chỉ ra rằng nếu ngôn ngữ ổn định (chứa các câu gồm các từ với cấu trúc thông dụng) thì công thức trên có thể biến đổi thành:

$$H(L) = - \lim_{m \to \infty} \frac{1}{m} \log p(w_1, w_2, ..., w_m)$$

Với công thức trên, ta có thể sử dụng công thức Bayes và xác suất của các n-gram để tính $p(w_1, w_2, ..., w_n)$:

$$H(L) = - \lim_{m \to \infty} \frac{1}{m} \log [p(w_n|w_1 w_2 .. w_{n-1}) * p(w_{n+1}|w_2 w_3 .. w_n) * ... * p(w_m |w_{m-n+1} ... w_{m-1})]$$

Công thức trên đã được biến đổi qua nhiều bước với các xấp xỉ gần đúng, do vậy để tăng tính chính xác khi sử dụng độ đo entropy thì câu kiểm tra cần phải đủ dài và tổng quát (phân tán rộng) để tránh tập trung vào các xác suất lớn (chỉ chứa các cụm thông dụng).

Các bước biến đổi gần đúng công thức trên khiến giá trị H(L) tính theo công thức cuối cùng sẽ lớn hơn giá trị H(L) gốc. Do vậy, khi tính H(L) của các mô hình ngôn ngữ khác nhau trên ngôn ngữ L, mô hình nào cho H(L) nhỏ hơn thì mô hình ngôn ngữ đó thể hiện chính xác ngôn ngữ L hơn.

1.5.2 Perplexity – Độ hỗn loạn thông tin:

Độ hỗn loạn thông tin (**perplexity)** cũng được dùng làm thước đo để đánh giá độ chính xác của một mô hình ngôn ngữ. Trong mô hình ngôn ngữ, độ hỗn loạn thông tin của một văn bản với từ "cái" thể hiện số từ có thể đi sau từ "cái". Độ hỗn loạn thông tin của một mô hình ngôn ngữ nói chung, có thể hiểu đơn giản là số lựa chọn từ trung bình mà mô hình ngôn ngữ phải đưa ra quyết định. Như vậy, độ hỗn loạn thông tin càng thấp, thì độ chính xác của mô hình ngôn ngữ càng cao.

Độ hỗn loạn thông tin có thể tính theo công thức:

$$P(L) = 2^{H(L)}$$

Ví duL dãy kí tự a, b,…, z có perplexity là 26 còn bảng mã ASCII có perplexity là 256.

1.5.3Error rate – Tỉ lệ lỗi:

Người ta thường sử dụng độ đo entropy và perplexity để so sánh độ chính xác của các mô hình ngôn ngữ khi xây dựng một mô hình ngôn ngữ tổng quát. Trong các bài toán cụ thể, người ta sử dụng tỉ lệ lỗi để so sánh độ chính xác của các mô hình ngôn ngữ.

Soát lỗi chính tả: xét tỉ lệ giữa số lỗi phát hiện sai hoặc không phát hiện được trên tổng số lỗi có trong văn bản.

Phân đoạn từ: xét tỉ lệ giữa từ phân đoạn sai trên tổng số từ có trong văn bản

Bỏ dấu tự động: xét tỉ lệ giữa số từ bị bỏ dấu nhầm trên tổng số từ có trong văn bản

Tỉ lệ lỗi thấp chứng tỏ mô hình ngôn ngữ hiệu quả. Việc sử dụng tỉ lệ lỗi để đánh giá đưa lại kết quả chính xác nhất khi muốn chọn lựa mô hình ngôn ngữ phù hợp để giải quyết bài toán cụ thể. Tỉ lệ lỗi thường tỉ lệ thuận với giá trị entropy nhưng đôi khi mức độ tăng/giảm của tỉ lệ lỗi và entropy không đều.

Chương 2: Tổng quan về Hadoop MapReduce

Trong chương này sẽ trình bày các kiến thức cơ bản về Hadoop và MapReduce. Trình bày về kiến trúc và cơ chế hoạt động của Hadoop và MapReduce.

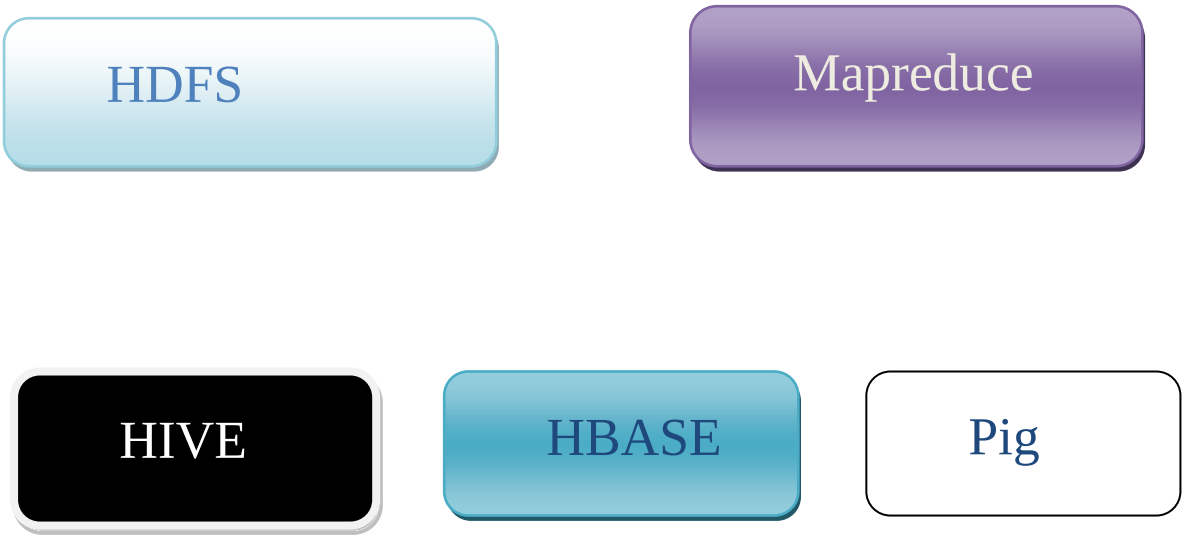
2.1 Hadoop

Apache Hadoop là framework mã nguồn mở [9].Nó dựa trên Java và sử dụng hệ thống tệp phân tán Hadoop (HDFS).Hadoop hiện thực mô hình Mapreduce, đây là mô hình mà ứng dụng sẽ được chia nhỏ ra thành nhiều phân đoạn khác nhau và các phần này sẽ được chạy trên nhiều node khác nhau.

2.2 Các thành phần của Hadoop

Trong phần này sẽ trình bày kiến trúc tổng quan của Hadoop. Hadoop bao gồm các thành phần sau [11]:

- HDFS – Hệ thống tệp phân tán
- MapReduce: Mô hình xử lý dữ liệu phân tán
- Hive: Kho dữ liệu phân tán, cung cấp SQL dựa trên ngôn ngữ truy vấn
- HBase: Cơ sở dữ liệu dựa trên cột phân tán
- Pig: Ngôn ngữ dòng dữ liệu và môi trường thực thi



Hình 2.1: Kiến trúc Hadoop

2.2.1 Kiến trúc hệ thống tệp phân tán

Giống như các hệ thống tệp khác, HDFS duy trì một cấu trúc cây phân cấp các tệp.Các tệp được lưu trữ bằng một hay nhiều Block. Mỗi block có kích thước là 64MB và có một Id riêng.

HDFS có một kiến trúc master/slave, trên một cluster chạy HDFS, có hai loại node là Namenode và Datanode. Một cluster có duy nhất một Namenode và có một hay nhiều Datanode.

Namenode đóng vai trò là master, chịu trách nhiệm duy trì thông tin về cấu trúc cây phân cấp các tệp, thư mục của hệ thống tệp và các metadata khác của hệ thống tệp. Cụ thể, các Metadata mà Namenode lưu trữ gồm có:

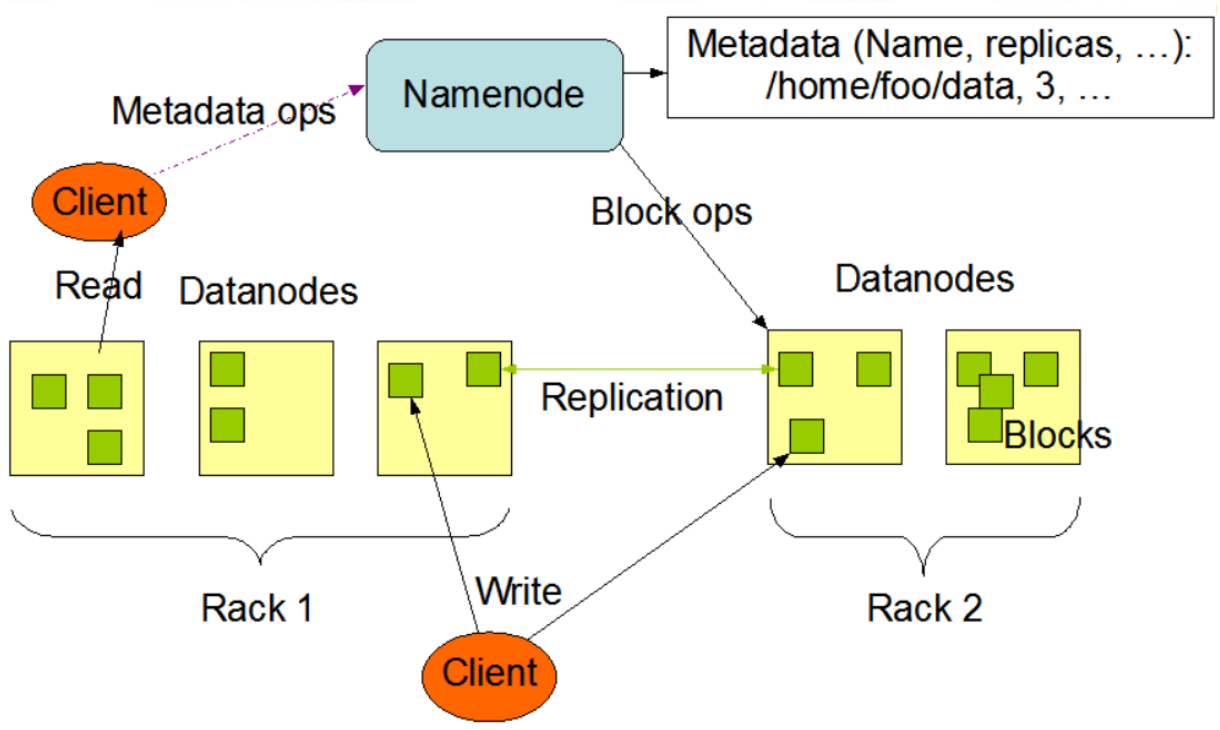
* File System Namespace: là hình ảnh cây thư mục của hệ thống file tại một thời điểm nào đó. File System namespace thể hiện tất các tệp, thư mục có trên hệ thống tệp và quan hệ giữa chúng.

* Thông tin để ánh xạ từ tên file ra thành danh sách các block: với mỗi tệp, ta có một danh sách có thứ tự các block của tệp đó, mỗi Block đại diện bởi Block ID.

* Nơi lưu trữ các block: các block được đại diện một Block ID. Với mỗi block ta có một danh sách các DataNode lưu trữ các bản sao của block đó.

Datanote: Lưu trữ nội dụng các tệp bằng các blocks. Mỗi block của cùng một tệp sẽ lưu trên các DataNode khác nhau.

Kiến trúc của HDFS như sau



Hình 2.2: Kiến trúc của HDFS

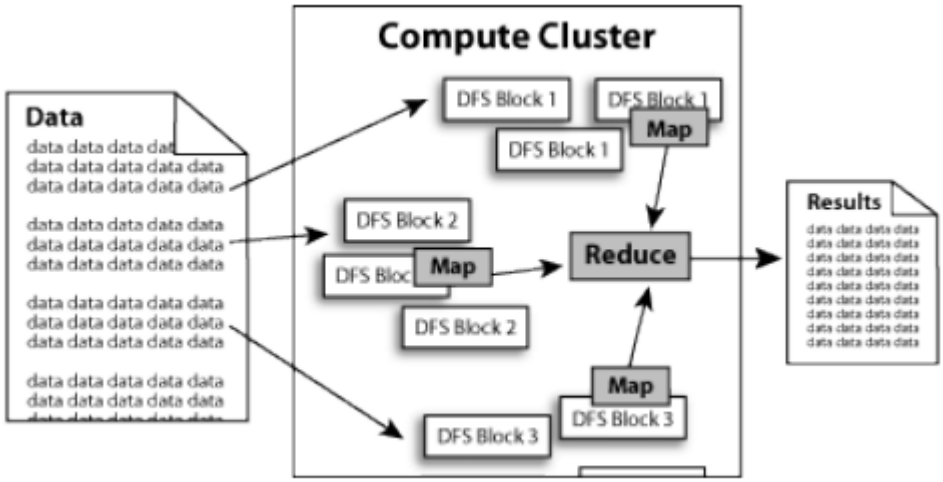
2.3 Mapreduce

MapReduce được thiết kế bởi Google như một mô hình lập trình xử lý tập dữ liệu lớn song song, thuật toán được phân tán trên 1 cụm. Mặc dù, ban đầu MapReduce là công nghệ độc quyền của Google, nhưng trong thời gian gần đây nó đã trở thành thuật ngữ tổng quát hóa.

Chương trình MapReduce chạy với 2 giai đoạn sau:

1. Giai đoạn Map
2. Giai đoạn Reduce.

Mapreduce hoạt động như sau: Đầu tiên các tệp đầu vào được chia nhỏ ra thành các khối nhỏ hơn có tên là FileSplits và hàm Map tạo ra các phần song song với từng task trên các FileSplit.



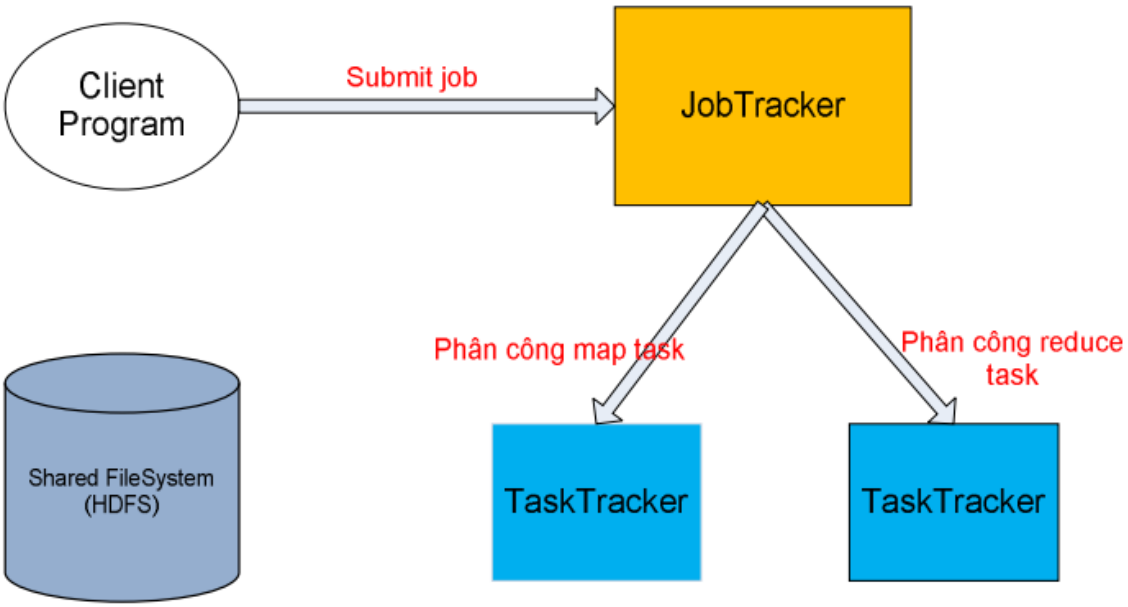
Hình 2.3: Mô hình Mapreduce

Đầu vào và đầu ra của một Map-reduce job:

(đầu vào) <k1,v1> -> map -><k2, v2> -> combine -><k2, v2> -> reduce -><k3, v3> (đầu ra)

Các file đầu vào được coi là một cặp khóa / giá trị và lập trình viên dùng một hàm Map để xử lý các cặp khóa/giá trị để tạo ra một tập các cặp khóa / giá trị trung gian. Khi hàm Map kết thúc đầu ra được đưa tới các Partitioner thường là một hàm băm, tất cả các cặp có cùng khóa sẽ được tập hợp cùng nhau. Sau khi các cặp trung gian được tạo ra một hàm Combine sẽ được gọi để reduce trong mỗi nút để tăng tốc độ xử lý. Sau đó một hàm Reduce trộn tất cả các giá trị cùng key và ghi vào các tệp đầu ra. Map và Reduce làm việc độc lập trên các khối dữ liệu. Kết quả đầu ra sẽ là một tệp trên mỗi reduce, và Hadoop lưu các đầu ra trên HDFS

2.3.1Kiến trúc của Mapreduce



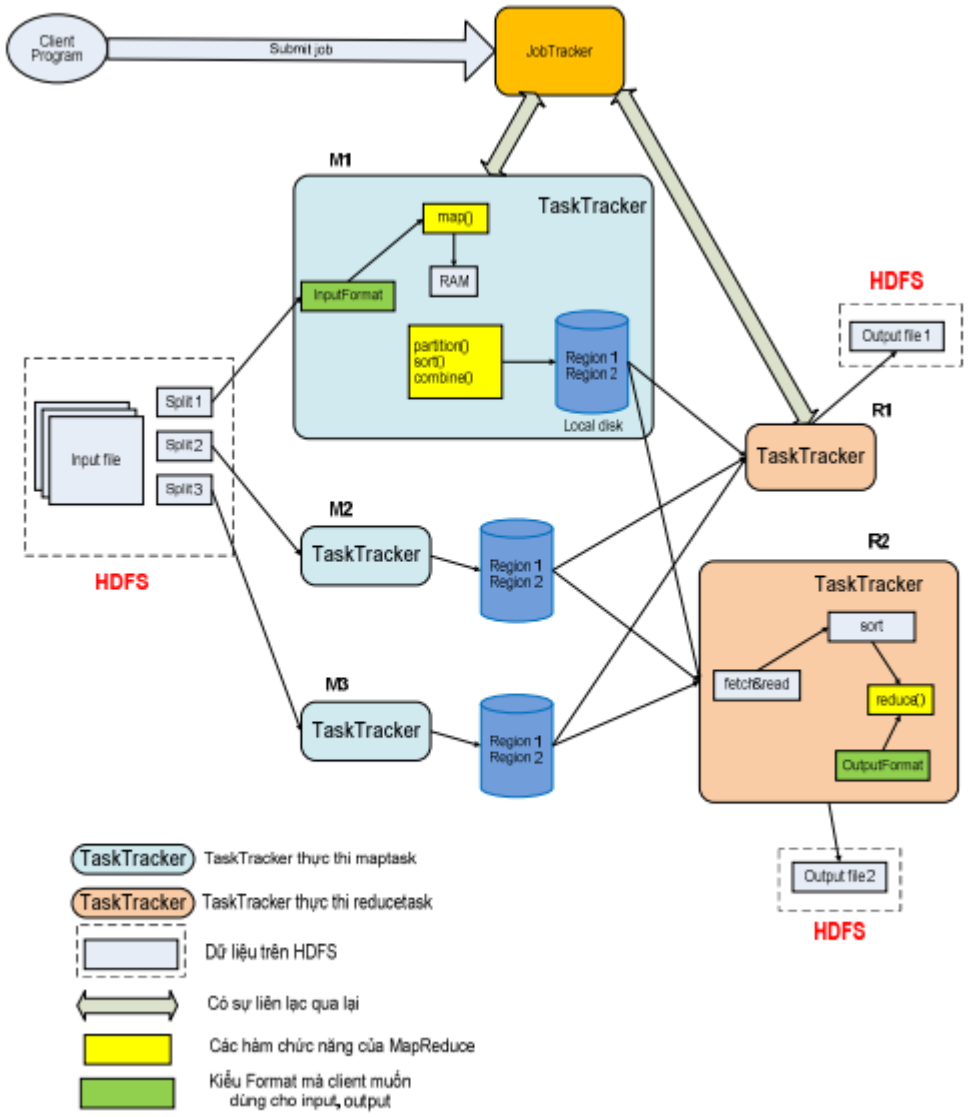
Hình 2.4: Kiến trúc MapReduce

Client Program*:* Chương trình HadoopMapReduce mà client đang sử dụng và tiến hành chạy một MapReduce Job.

JobTracker: Tiếp nhận job và đảm nhận vai trò điều phối job này, nó có vai trò như bộ não của Hadoop MapReduce. Sau đó, nó chia nhỏ job thành các task, tiếp theo sẽ lên lịch phân công các task (map task, reduce task) này đến các tasktracker để thực hiện. Kèm theo vai trò của mình, JobTracker cũng có cấu trúc dữ liệu riêng của mình để sử dụng cho mục đích lưu trữ, ví dụ như nó sẽ lưu lại tiến độ tổng thể của từng job, lưu lại trang thái của các TaskTracker để thuận tiện cho thao tác lên lịch phân công task, lưu lại địa chỉ lưu trữ của các output của các TaskTracker thực hiện maptask trả về.

TaskTracker*:* Đơn giản nó chỉ tiếp nhận maptask hay reducetask từ JobTracker để sau đó thực hiện. Và để giữ liên lạc với JobTracker, Hadoop Mapreduce cung cấp cơ chế gửi heartbeat từ TaskTracker đến JobTracker cho các nhu cầu như thông báo tiến độ của task do TaskTracker đó thực hiện, thông báo trạng thái hiện hành của nó

2.3.2 Cơ chế hoạt động



Hình 2.5: Cơ chế hoạt động của MapReduce

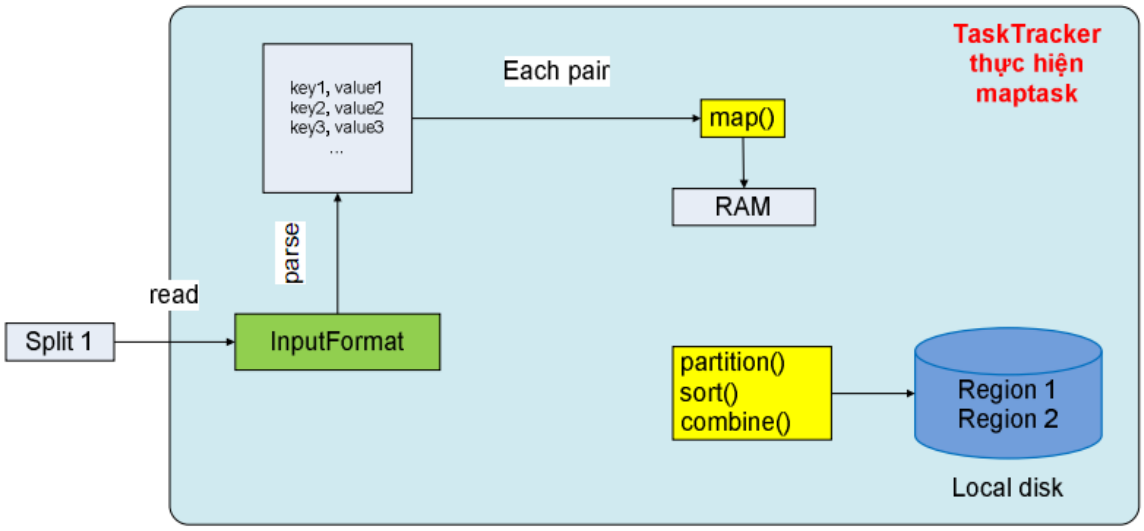
Đầu tiên chương trình client sẽ yêu cầu thực hiện job và kèm theo là dữ liệu đầu vào tới JobTracker. JobTracker sau khi tiếp nhận job này, nó sẽ thông báo ngược về chương trình client tình trạng tiếp nhận job. Khi chương trình client nhận được thông báo nếu tình trạng tiếp nhận hợp lệ thì nó sẽ tiến hành phân rã dữ liệu đầu vào này thành các split (khi dùng HDFS thì kích thước một split thường bằng với kích thước của một đơn vị Block trên HDFS) và các split này sẽ được ghi xuống HDFS. Sau đó chương trình client sẽ gửi thông báo đã sẵn sàng để JobTracker biết rằng việc chuẩn bị dữ liệu đã thành công và hãy tiến hành thực hiện job.

Khi nhận được thông báo từ chương trình client, JobTracker sẽ đưa job này vào một stack mà ở đó lưu các job mà các chương trình client yêu cầu thực hiện. Tại một thời điểm JobTracker chỉ được thực hiện một job. Sau khi một job hoàn thành hay bị

block, JobTracker sẽ lấy job khác trong stack này (FIFO) ra thực hiện. Trong cấu trúc dữ liệu của mình, JobTrack có một job scheduler với nhiệm vụ lấy vị trí các split (từ HDFS do chương trình client tạo), sau đó nó sẽ tạo một danh sách các task để thực thi. Với từng split thì nó sẽ tạo một maptask để thực thi, mặc nhiên số lượng maptask bằng với số lượng split. Còn đối với reduce task, số lượng reduce task được xác định bởi chương trình client. Bên cạnh đó, JobTracker còn lưu trữ thông tin trạng thái và tiến độ của tất cả các task.

Hình 2.6: Mối quan hệ giữa JobTracker và Task Tracker

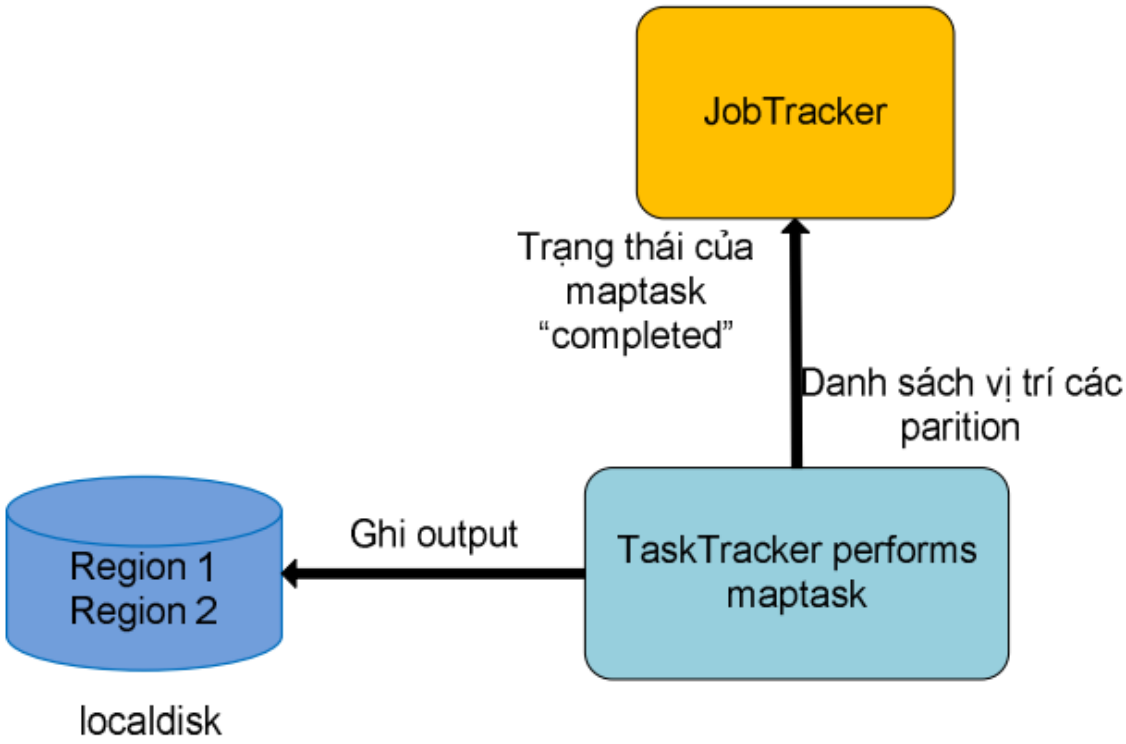
Ngay khi JobTracker khởi tạo các thông tin cần thiết để chạy job, thì bên cạnh đó các TaskTracker trong hệ thống sẽ gửi các heartbeat đến JobTracker. Hadoop cung cấp cho các TaskTracker cơ chế gửi heartbeat đến JobTracker theo chu kỳ thời gian nào đó, thông tin bên trong heartbeat này cho phép JobTrack biết được TaskTracker này có thể thực thi task hay. Nếu TaskTracker còn thực thi được thì JobTracker sẽ cấp task và vị trí split tương ứng đến TaskTracker này để thực hiện.



Hình 2.7: Mô hình Task Tracker

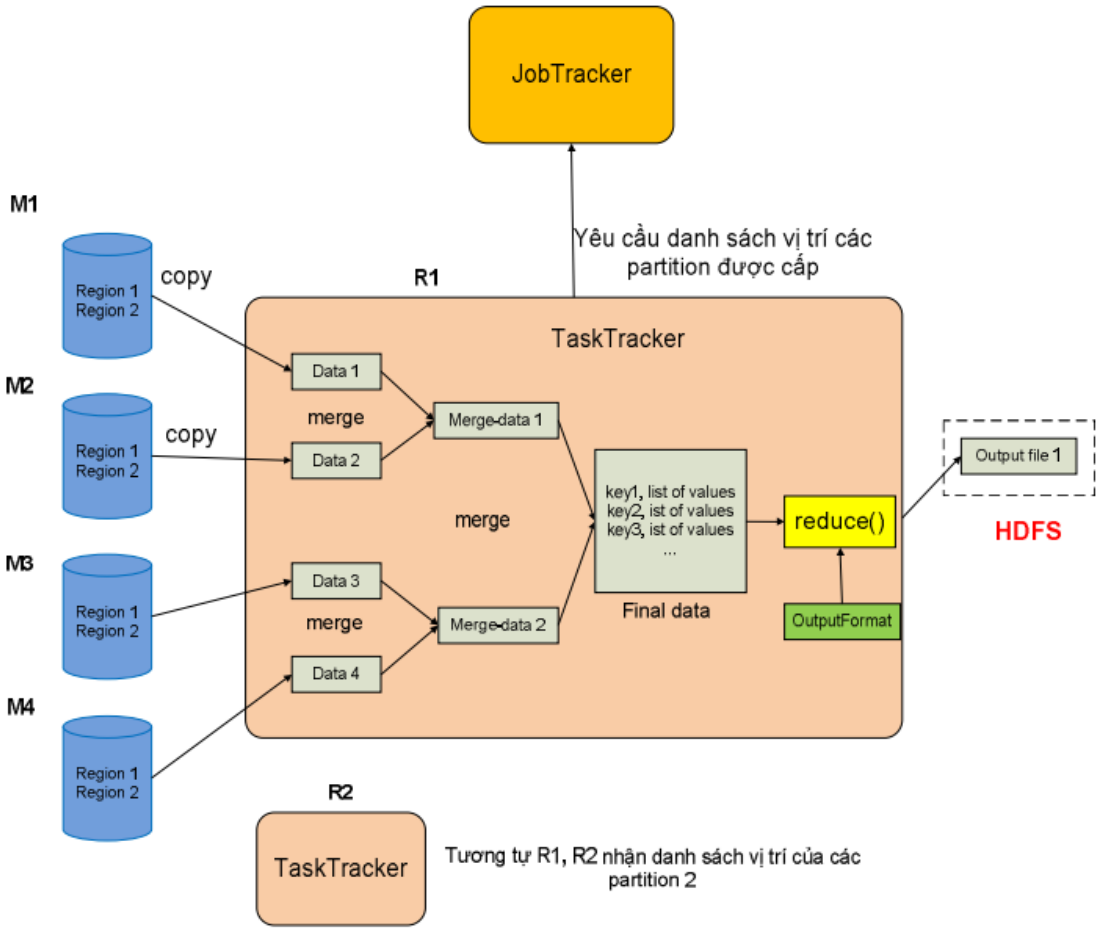
Khi một TaskTracker nhận thực thi maptask, kèm theo đó là vị trí của input split trên HDFS. Sau đó, nó sẽ nạp dữ liệu của split từ HDFS vào bộ nhớ, rồi dựa vào kiểu format của dữ liệu input do chương trình client chọn thì nó sẽ parse split này để phát

sinh ra tập các record, và record này có 2 trường: key và value. Cho ví dụ, với kiểu input format là text, thì tasktracker sẽ cho phát sinh ra tập các record với key là offset đầu tiên của dòng (offset toàn cục), và value là các ký tự của một dòng. Với tập các record này, tasktracker sẽ chạy vòng lặp để lấy từng record làm input cho hàm map để trả ra out là dữ liệu gồm intermediate key và value. Dữ liệu output của hàm map sẽ ghi xuống bộ nhớ chính, và chúng sẽ được sắp xếp trước ngay bên trong bộ nhớ chính



Hình 2.8: Mô hình JobTracker

Trước khi ghi xuống local disk, các dữ liệu output này sẽ được phân chia vào các partition (region) dựa vào hàm partition, từng partition này sẽ ứng với dữ liệu input của reduce task sau này. Và ngay bên trong từng partition, dữ liệu sẽ được sắp xếp (sort) tăng dần theo intermediate key, và nếu chương trình client có sử dụng hàm combine thì hàm này sẽ xử lý dữ liệu trên từng partition đã sắp xếp rồi. Sau khi thực hiện thành công maptask thì dữ liệu output sẽ là các partition được ghi trên local, ngay lúc đó TaskTracker sẽ gửi trạng thái completed của maptask và danh sách các vị trí của các partition output trên localdisk của nó đến JobTracker



Hình 2.9: Cơ chế hoạt động của JobTracker

Sau khi nạp thành công tất cả các region thì TaskTracker sẽ tiến hành merge dữ liệu của các region theo nhiều đợt mà các đợt này được thực hiện một cách đồng thời để làm gia tăng hiệu suất của thao tác merge. Sau khi các đợt merge hoàn thành sẽ tạo ra các file dữ liệu trung gian được sắp xếp. Cuối cùng các file dữ liệu trung gian này sẽ được merge lần nữa để tạo thành một file cuối cùng. TaskTracker sẽ chạy vòng lặp để lấy từng record ra làm input cho hàm reduce, hàm reduce sẽ dựa vào kiểu format của output để thực hiện và trả ra kết quả output thích hợp. Tất cả các dữ liệu output này sẽ được lưu vào một file và file này sau đó sẽ được ghi xuống HDFS.

2.4Ưu điểm của Hadoop

Hadoop framework giúp người sử dụng nhanh chóng kiểm tra các hệ thống phân tán đây được xem là phương pháp phân phối dữ liệu và công việc xuyên suốt các máy trạm nhờ vào cơ chế xử lý song song của các lõi CPU

Hadoop được thiết kế phát hiện và xử lý các lỗi ở lớp ứng dụng mà không dựa vào cơ chế chịu lỗi FTHA (Fault tolerance and high availability)

Linh hoạt trong việc thêm và gỡ bỏ từ các cluster, không bị ngắt quãng

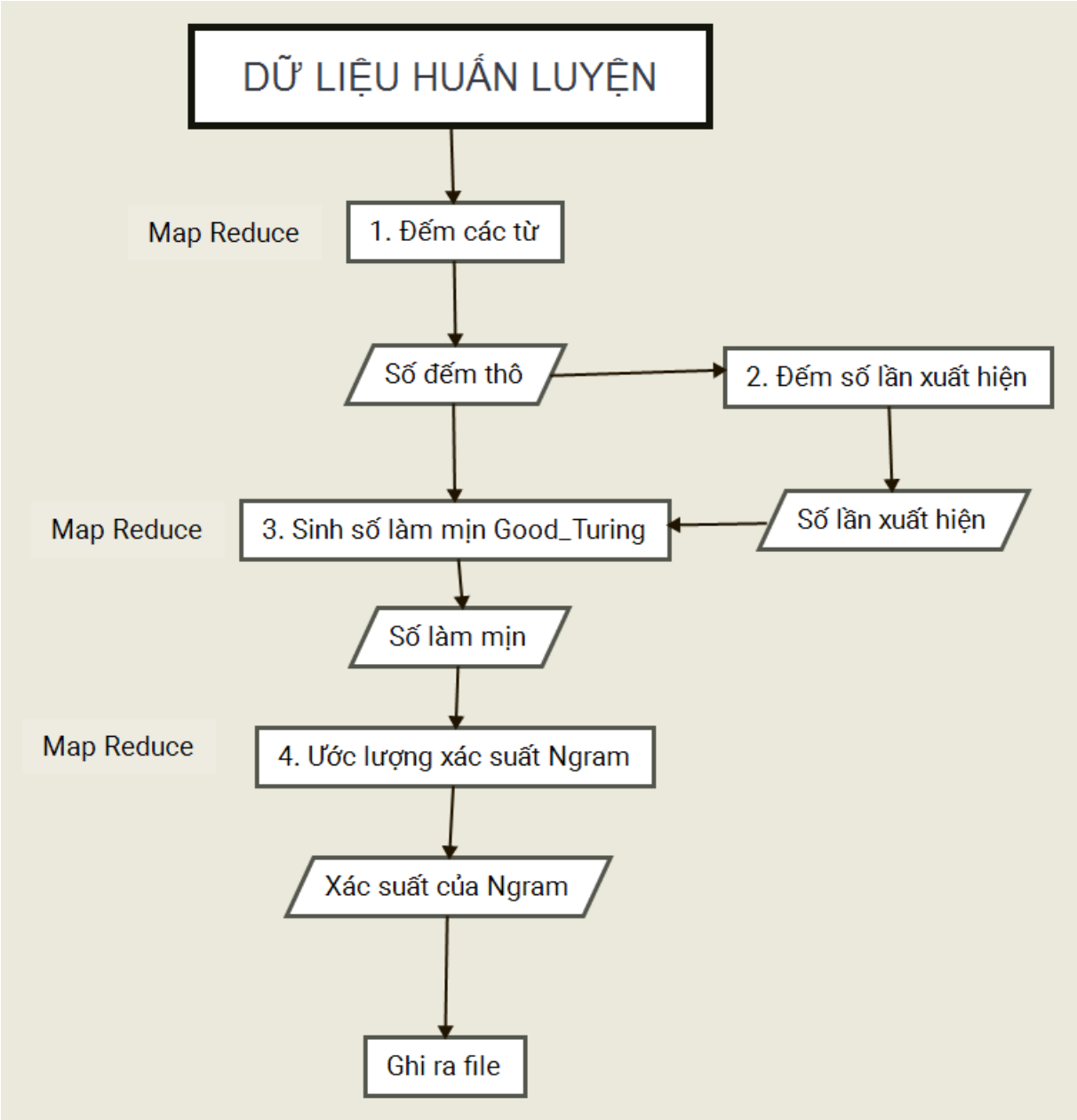
Khả năng tương thích trên tất cả các nền tảng do được phát triển trên java

Chương 3:Ước lượng mô hình ngôn ngữ với Mapreduce

Để ước lượng mô hình ngôn ngữ được chính xác thì cần phải sử dụng bộ dữ liệu huấn luyện lớn. Bộ dữ liệu càng lớn thì mô hình ngôn ngữ càng chính xác [13]. Như vậy sẽ cần bộ nhớ lưu trữ là rất lớn và một chương trình phải đủ nhanh để thực hiện. Hadoop và MapReduce là một công cụ để xử lý dữ liệu khổng lồ. Hadoop và MapReduce [10] là một lựa chọn tốt cho bài toán xây dựng mô hình ngôn ngữ với dữ liệu lớn.

Quá trình phân tích dữ liệu tập huấn trong Google được chia thành ba bước: Chuyển các từ thành các id, sinh ra các n-gram trên từng câu và tính toán xác xuất của các n-gram. Sử dụng ước lượng làm mịn Good-Turing và một số bước phụ thêm bao gồm tính toán số lượng của các n-gram, lưu các số lượng này trong HDFS và sau đó lấy dữ liệu để điều chỉnh các số lượng.

Hình sau mô tả quá trình thực hiện.



Hình 3.1: Quá trình xử lý

3.1 Đếm các từ

Bước đầu tiên là phân tích dữ liệu huấn luyện, tìm tất cả các n-gram và số lượng của chúng [12]. Hàm map sẽ đọc từng dòng trong dữ liệu đầu vào. Khóa là docid, và giá trị là văn bản. Trong từng dòng sẽ được phân vào tất cả các 1-gram, 2-gram, 3-gram. Những n-gram là các khóa trung gian và giá trị là 1. Sau đó một hàm Combine đếm tất cả các giá trị cho cùng khóa trong các Map task. Và một hàm reduce giống như combiner tập hợp tất cả các đầu ra từ combiner tính tổng giá trị cho cùng một khóa. Khóa cuối cùng là giống với đầu ra của hàm map là các n-gram và giá trị là số

các n-gram trong tập dữ liệu huấn luyện. một partitioner dựa trên hashcode của 2 từ đầu được sử dụng. Hàm map và reduce như sau:

```
map(LongWritabe docid, Text line, OutputCollector<Text, IntWritable>
output) {
1:    words[] = line.split(blank space or punctuation)
2:    for i=1..order
3:    for k = 0..(words.length - i)
4:    output.collect(words[k..(k+i-1)], 1)
}
```

Hàm Reduce như sau:

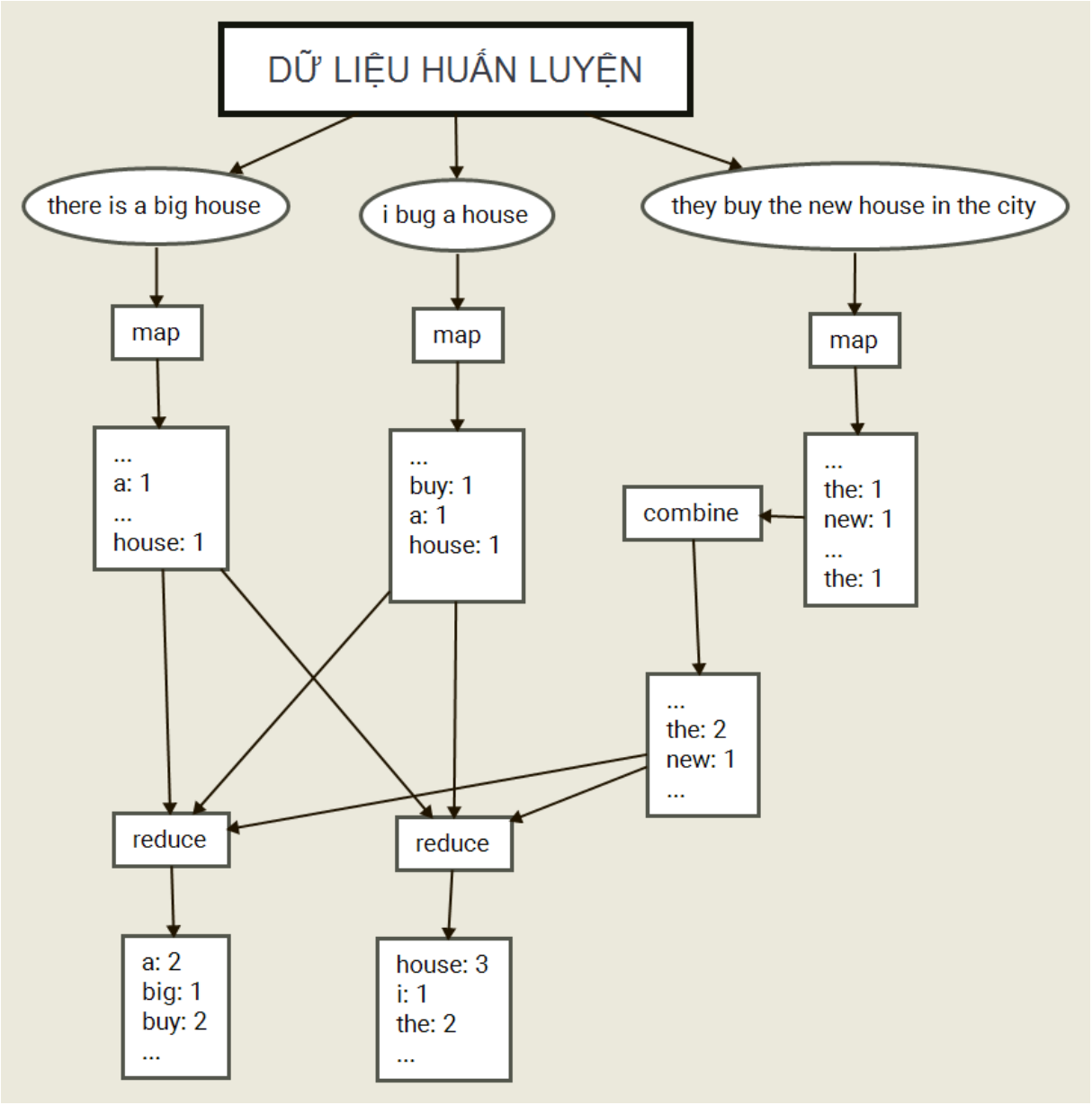
```
reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output ) {
1:    int sum=0;
2:    while (values.hasNext())
3:    sum += values.next().get();
4:    if sum > prune
5:    output.collect(key, sum);
}
```

Dưới đây là ví dụ cho quá trình đếm số lượng các từ với tập dữ liệu huấn luyện như sau:

There is a big house

I buy a house

They buy the new house in the city



Hình 3.2: Đếm các từ

Hình trên mô tả quá trình của các hàm map – combine – reduce cho tập huấn luyện trên. Hàm combine bắt đầu sau hàm map, vì vậy nó kế thừa các cặp khóa /giá trị từ các task map trước của nó. Đầu ra từ các hàm reduce là các số lượng từ cũng như là khóa được lưu.

Bởi vì những bước này sinh ra tất cả các n-gram nó có thể tập hợp tất cả các số lượng của các 1-gram, 2-gram. Những số lượng này là cần thiết cho kỹ thuật làm mịn. Vì vậy ở đây chỉ tập hợp số lượng các 1-gram cho kỹ thuật làm mịn Good-Turing. Nó cũng dễ dàng tập hợp tất cả các 2-gram hoặc 3-gram cần thiết cho kỹ thuật làm mịn Kneser-Ney.

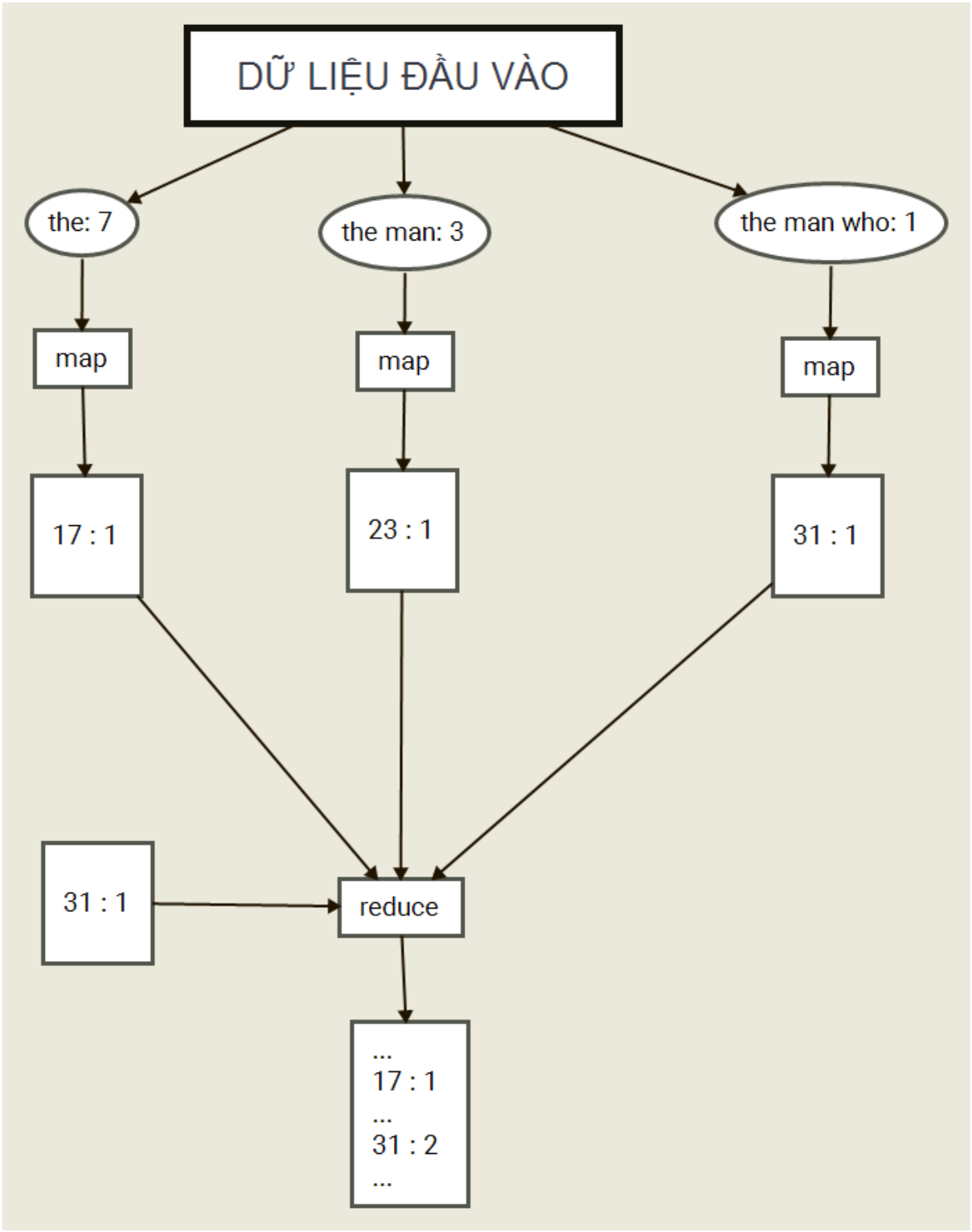
```
enum MyCounter {INPUT_WORDS};
reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output,
Reporter reporter) {
.....
if sum>prune
output.collect(key, sum);
if key is unigram
reporter.incrCounter(MyCounter.INPUT_WORDS, 1);
}
```

3.2Đếm số lần xuất hiện (Generate count of counts)

Kỹ thuật làm mịn Good – Turing được dựa trên việc đếm số lượng của các n-gram. Vì vậy cần thiết phải tập hợp tất cả số lượng của số các 1-gram, 2-gram, 3- gram cho đến n-gram. Để làm được việc này thì tất cả các dòng số lượng được đưa vào một MapReduce mới. Cho mỗi n-gram, hàm map sẽ đưa ra một số của dòng số lượng với thứ tự của n-gram và khóa đưa ra sẽ có dạng <n-gram – oder raw-counts> và giá trị là <count-of-counts>. Hàm combine và reduce sẽ ghép tất cả các số lần xuất hiện cùng với khóa. Số lượng đưa ra nhỏ thường là một file có thể lưu đủ các số của số lượng. Hàm map như sau:

```
// input key is ngram, input value is the raw counts
publicvoidmap(Text key, IntWritable value,
OutputCollector<IntWritable, IntWritable> output){
1:    words[] = toStringArray(key);
2:    String combine = words.length + value.toString();
3:    output.collect(toText(combine), one);
}
```

Hàm combine và reduce giống như bước trên. Sau đây là một ví dụ cho MapReduce.



Hình 3.3: Đếm số lượng

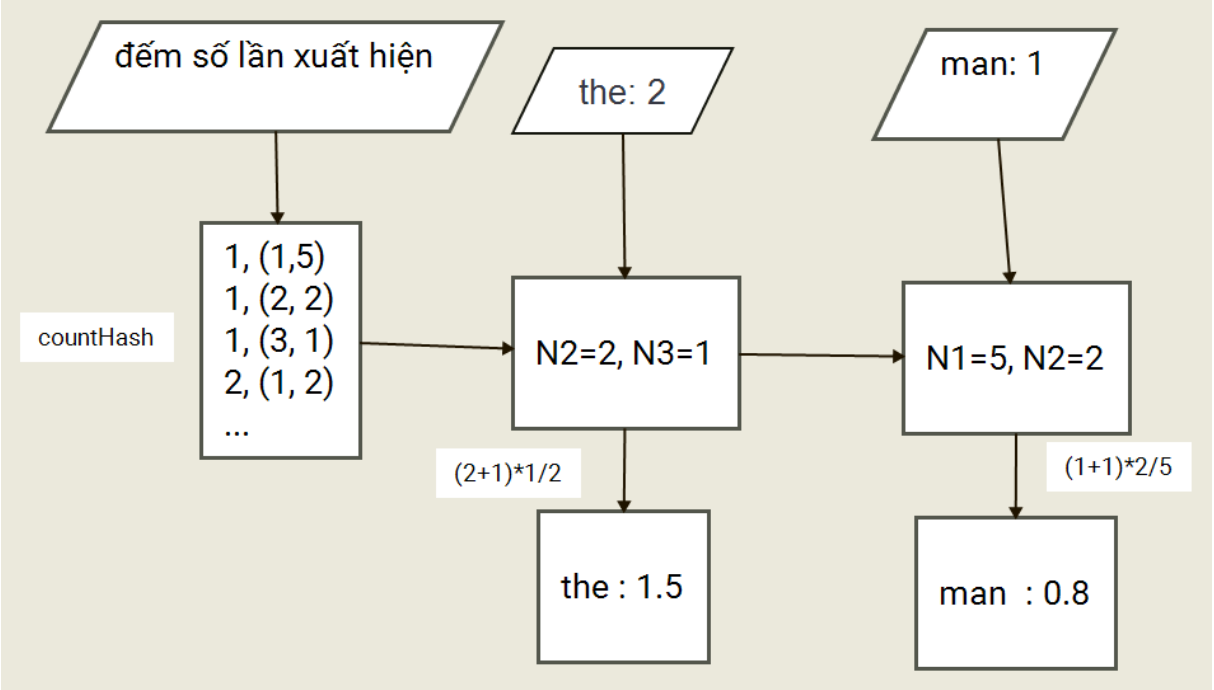
3.3Sinh số làm mịn Good-Turing

Với công thức làm mịn Good-Turing chúng ta có thể ước lượng số được làm mịn cho mỗi n-gram. Trong bước này các dữ liệu đầu vào vẫn là các n-gram và các số đếm của chúng, mỗi hàm map sẽ đọc trong từng dòng số của các số, lưu tất cả dữ liệu trong cấu trúc HashMap và tính số được làm mịn. Theo công thức là:

$$r^* = (r+1)\frac{N_{r+1}}{N_r}$$

Nếu có thể tìm cả N_{r+1} và N_r trong HashMap thì công thức trên có thể áp dụng trực tiếp. Ngược lại sẽ thử tìm trong số của số lượng gần nhất, nếu không thể tìm được N_{r+1} thì sẽ thử tìm N_{r+2}, N_{r+3}, N_{r+4}, … Trong phần thử nghiệm chúng tôi đã tìm trong hầu hết 5 số N_{r+1}, …, N_{r+5}. Nếu không tìm thấy bất kỳ những số này, thì sẽ dùng số đếm thô để thay thế. Trong tình huống này thì số đếm thô này rất rộng, có nghĩa là n-gram có xác suất liên quan cao nhưng chúng ta không thể điều chỉnh được số lượng. Với mỗi n-gram việc xử lý làm mịn cần thiết chỉ một lần, vì vậy thực tế không cần bất kỳ combine hoặc reduce cho số làm mịn.

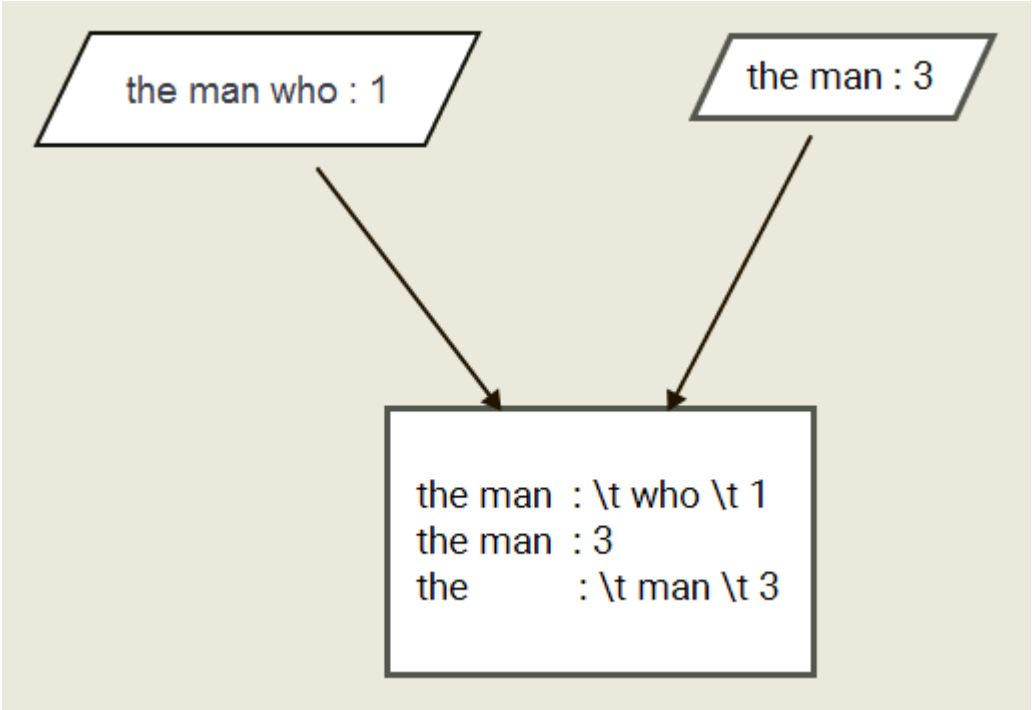
Hình sau sẽ đưa ra một ví dụ của phương pháp Good-Turing



Hình 3.4: Làm mịn Good-Turing

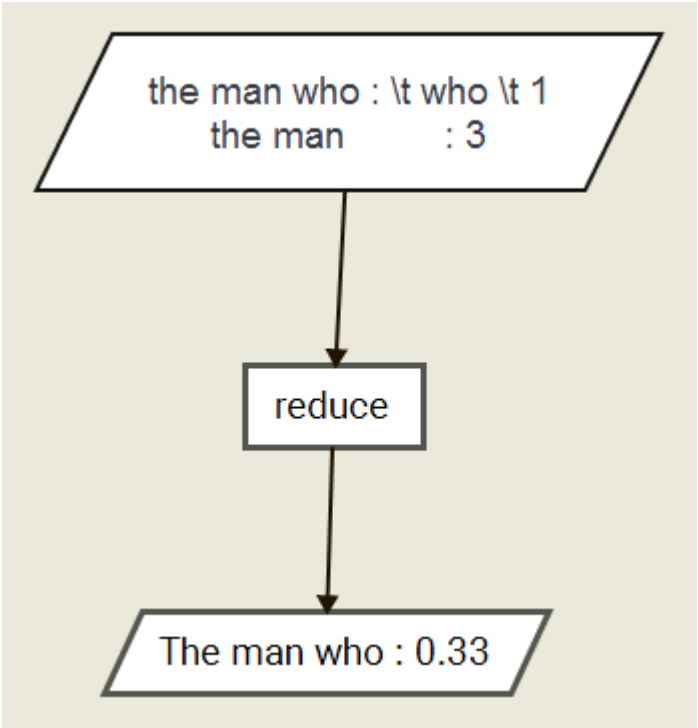
3.4Ước lượng xác suất n-gram

Để ước lượng xác suất của một n-gram $w_1,w_2,\ldots,w_n$, chúng ta cần số lượng của $w_1,\ldots,w_n$ và $w_1\ldots w_{n-1}$. Bởi vì một chương trình MapReduce mỗi map hoặc reduce đang làm việc dựa trên một khóa, Thử nghiệm sẽ sử dụng chuỗi $w_1,w_2,\ldots,w_{n-1}$ như là khóa và tổ hợp từ w_n với số lượng của từ đó là giá trị. Việc này hoàn thành trong hàm map ở bước trước. Hình sau là một ví dụ đưa ra văn bản với định dạng từ hiện tại



Hình 3.5: Quá trình sinh đoạn văn với từ hiện tại

Sau khi làm mịn Good-Turing một vài số có thể khá nhỏ, vì vậy xác suất có thể lớn hơn 1. Trong trường hợp này chúng ta cần điều chỉnh nó xuống 1. Cho một mô hình back-off chúng ta sử dụng một khối đơn giản được cung cấp bởi Google, trong đó số back-off được đặt là 0.4. Số 0.4 được chọn dựa trên kinh nghiệm và được phát triển trên các lựa chọn của các bước trước. Nếu muốn ước lượng số back-off ngẫu nhiên cho mỗi n-gram thì sẽ phải thực hiện nhiều bước hơn.



Hình 3.6: Ước lượng xác suất

Trong bước này chúng ta sẽ lấy tất cả các xác suất cho n-gram và với số back-off chúng ta có thể ước lượng xác suất cho kiểm thử các n-gram dựa trên truy vấn. Vì vậy bước tiếp theo là một bước quan trọng để lưu trữ những xác suất này trong môi trường phân tán.

3.5Sinh bảng Hbase

Hbase có thể sử dụng khi đưa đầu vào hoặc ra vào trong các Hadoop và MapReduce. Các sửa đổi là cần thiết bởi vì bảng Hbase được viết theo từng dòng, mỗi khi chúng ta cần sinh ra một khóa với nội dung là các cột. Có một vài lựa chọn ở đây, hoặc đơn giản là cho mỗi n-gram trên một dòng, hoặc nhiều dòng được cấu trúc dựa trên từ hiện tại và nội dung. Có hai vấn đề lớn cần phải quan tâm là tốc độ viết/truy vấn và kích thước bảng lưu trữ.

3.5.1 Cấu trúc dựa trên n-gram

Một cấu trúc khởi tạo là rất đơn giản tương tự như định dạng đầu ra của đoạn văn bản. Mỗi n-gram được lưu trữ trong một dòng riêng biệt, vì vậy bảng có cấu trúc phẳng với một cột. cho mỗi dòng, khóa là n-gram và cột lưu trữ xác xuất của chúng. Bảng 3.1 là một ví dụ đơn giản của cấu trúc này. Cấu trúc này dễ dàng để thực thi và bảo hành,

Khóa	Cột
a	0.11
a big	0.67
a big house	1.0
buy	0.11
buy a	0.67
…	…

Bảng 3.1: Cấu trúc bảng dựa trên n-gram.

3.5.2 Cấu trúc dựa trên từ hiện tại

Cho tất cả các n-gram $w_1,w_2,\ldots,w_n$ cùng chia sẻ một từ hiện tại w_n chúng ta có thể lưu trữ chúng trong một dòng với cùng một khóa w_n. Tất cả các ngữ cảnh có khả

năng xảy ra sẽ được lưu trữ vào những cột riêng biệt, tên cột được đặt theo định dạng <column family: context>. Bảng 3.2 là một ví dụ. Bảng này là một bảng có cấu trúc cột. Cho mỗi từ với rất nhiều ngữ cảnh, dòng có thể là khá dài, trong đó với một vài từ với ngữ cảnh ít hơn thì dòng có thể sẽ ngắn hơn. Trong bảng này chúng ta sẽ giảm bớt số của các dòng và mở rộng tất cả các ngữ cảnh vào các cột riêng biệt. Vì vậy thay vì một cột đơn, chúng ta có nhiều cột nhỏ. Từ khung nhìn của cơ sở dữ liệu phân tán, dữ liệu được lưu trữ thưa thớt nhưng từ khung nhìn của cấu trúc dữ liệu nó vẫn khá là gần nhau. Ví dụ nếu chúng ta có 2 từ hiện tại trong 2 dòng và cả 2 đều có cùng ngữ cảnh hoặc có cùng xác suất cho một vài ngữ cảnh, chúng ta sẽ lưu trữ chúng riêng biệt. Những kết quả này trong nhiều cột có cấu trúc tương tự điều này có nghĩa là một vài loại sẽ bị trùng lặp.

Chúng ta cũng cần tập hợp các xác xuất của unigram cho mỗi từ hiện tại và lưu trữ chúng trong một cột riêng.

Key	Column family: label					
	gt:unigram	gt:a	gt:a big	gt:i	gt:buy	…
a	0.15				0.667	
big	0.057	0.667				
house	0.3	0.667	1.0			
buy	0.15			1.0		
…	…	…	…	…	…	…

Bảng 3.2: Cấu trúc bảng dựa trên từ

3.5.3 Cấu trúc dựa trên đoạn văn

Tương tự như cấu trúc dựa trên từ hiện tại, chúng ta có thể sử dụng một đoạn $w_1,w_2,\ldots,w_{n-1}$ như một khóa của một dòng, và lưu trữ tất cả các khả năng từ w_n theo sau trong các cột khác với định dạng <column family: word>. Bảng này sẽ có nhiều dòng được so sánh với bảng dựa trên từ nhưng sẽ ít hơn bảng dựa trên ngram. Cho tập dữ liệu lớn hoặc cho ngram cao, đoạn văn có thể khá dài mặt khác thì các cột có thể được giảm bớt. Các cột được chia ra bởi những từ khác nhau và cho tất cả các từ chỉ xuất hiện một lần, sự phân chia này là rất nhỏ chỉ một giá trị cột cho một phân chia. Nói chung nếu chúng ta có n 1-gram trong tập dữ liệu chúng ta sẽ có khoảng n cột trong bảng. Cho một tập dữ liệu huấn luyện gồm 100 triệu từ, số lượng 1-gram khoảng 30000 vì vậy bảng có thể rất thưa thớt. Ví dụ của cấu trúc này có trong bảng sau

key	Column family: label							
	gt:unigram	gt:a	gt:big	gt:i	gt:buy	gt:the	gt:house	
a	0.11		0.67				0.67	
a big							1.0	
buy	0.11	0.67				0.67		
buy a							1.0	
i	0.04				1.0			
…	…	…	…		…	…		

Bảng 3.3: Cấu trúc bảng dựa trên đoạn văn

Để hạn chế sự dư thừa này chỉ những khóa 1-gram được lưu trữ với xác suất của nó trong <gr: unigram> vì vậy chúng ta có thể lưu xác suất của “a big” trong <gr:unigram> .

Vì có thể có nhiều cột chỉ xuất hiện một lần và có cùng giá trị giống nhau thường thấy trong các ngram bậc cao có thể kết hợp các cột lại với nhau, giảm bớt hoặc chia tách cột.

3.5.4Cấu trúc dựa trên nửa ngram

Đối với hai cấu trúc trước đó, chúng ta có thể nhận được hoặc số lượng ngày càng lớn hay số hàng ngày càng lớn. Vì vậy có một sự tráo đổi giữa các hàng và các cột. Chúng ta có thể tổ hợp cấu trúc dựa trên các từ và cấu trúc dựa trên đoạn văn với nhau cân bằng số lượng giữa các hàng và các cột. Phương pháp là chia n-gram thành n/2-gram và sử dụng n/2 gram là giá trị dòng và n/2 gram là tên cột. Ví dụ cho mô hình 4-gram (w1, w2, w3, w4,) khóa của dòng là (w1 w1) và cột là <gt: w3 w4>. Ví dụ cho mô hình 4-gram như bảng sau

key	Column family: label						
	gt:unigram	gt:a	gt:big	gt:house	gt:big house	gt:new house	.
	0.11		0.67	0.67			
big				1.0		0.01	
buy	0.11	0.67					
buy a				1.0	0.04		
	0.04				1.0		
	…	.	..	…	…		

Bảng 3.4: Cấu trúc bảng dựa trên nửa ngram

Cho những ngram bậc cao hơn, cấu trúc này có thể giảm bớt nhiều dòng và thêm chúng vào các cột. Về mặt lý thuyết chi phí tách ngram thành các từ - đoạn và đoạn – từ là giống nhau nhưng n/2gram - n/2gram sẽ đòi hỏi sự phân tích nhiều hơn.

3.5.5Cấu trúc dựa trên số nguyên

Thay vì lư trữ tất cả các chuỗi chúng ta có thể chuyển tất cả các từ thành các số nguyên và lưu các số nguyên vào trong bảng. Bước mở rộng là cần thiết để chuyển mỗi 1-gram thành một số nguyên duy nhất và giữ sự chuyển đổi 1gram-integer này vào các hệ thống tệp phân tán. Ưu điểm của việc sử dụng các số nguyên là sẽ có kích thước nhỏ hơn so với lưu trữ chuỗi. Nhưng mặt khác chúng ta cần một quá trình để mã hóa/ giải mã để làm các việc chuyển đổi. Phương pháp này là một sự đánh đổi giữa thời gian tính toán và kích thước lưu trữ. Ngoài ra cấu trúc này có thể được kết hợp với các phương pháp trước đó để nén tốt hơn.

Dưới đây là ví dụ của phương pháp cấu trúc dựa trên số nguyên

1-gram	integer
a	1
big	2
house	3
buy	4

Key	Column family: label (gr:prob)
1	0.11
1 2	0.67
1 2 3	1.0
4	0.11
4 1	0.67

Bảng 3.5: Cấu trúc bảng dựa trên số nguyên

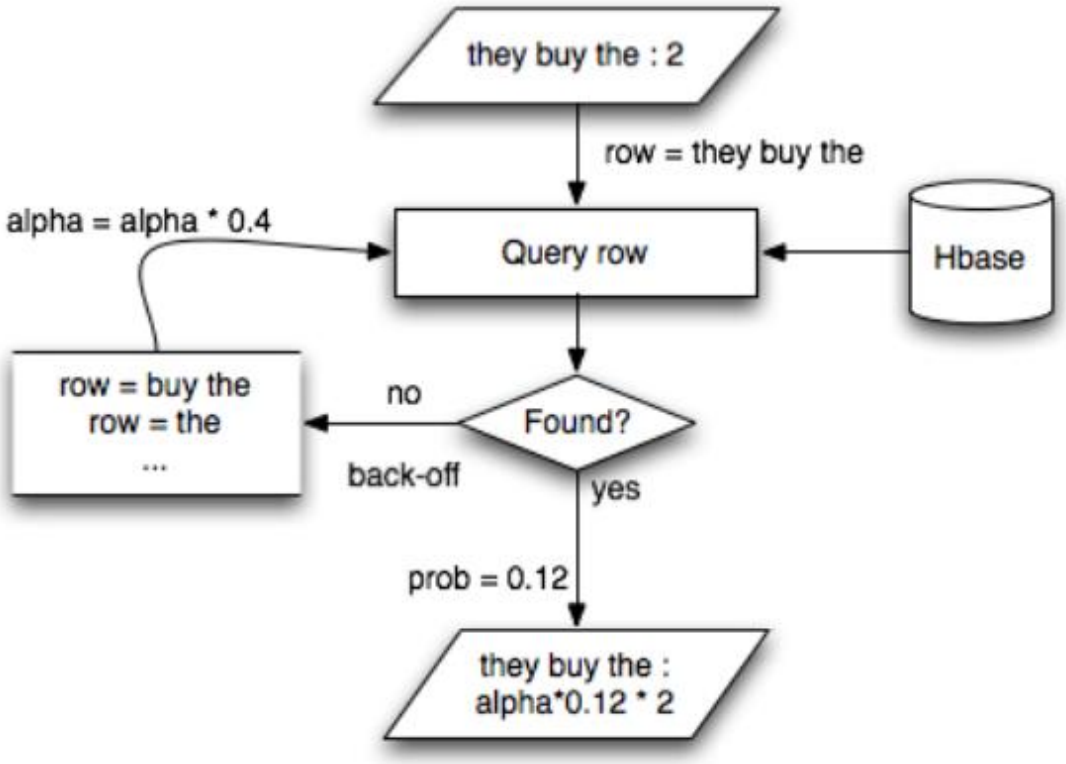
Lưu ý rằng nếu chúng ta lưu trữ “a big” là 12 nó có thể xung đột với từ khác có số là 12 trong hàm chuyển đổi. Vì vậy chúng ta phải thêm khoảng trống giữa các số giống như chuỗi ngram.

3.6Truy vấn trực tiếp

Quá trình xử lý tiếp theo là kiểm thử. Phương pháp back-off được thực hiện trong câu truy vấn ở đây. Dựa trên mô hình back-off cho mỗi n-gram thử nghiệm chúng ta cần truy vấn n-gram nếu không tìm thấy thì sẽ tìm trong n-1 gram cho đến khi chúng ta gặp 1-gram. Cho cấu trúc bảng khác chúng ta cần sinh ra các dòng khác nhau và đặt tên cho các cột. Ưu điểm của việc sử dụng MapReduce cho việc thử nghiệm là chúng ta có thể đưa nhiều chuỗi thử nghiệm vào HDFS và MapReduce có thể xử lý trong tất cả các chuỗi để đưa ra số đếm thô của chúng.

Phương thức này được gọi là truy vấn trực tiếp bởi vì nó truy vấn mỗi n-gram trực tiếp từ bảng Hbase, vì nhiều n-gram được kiểm thử thì thời gian cho việc thử nghiệm này cũng sẽ nhiều.

Hình 3.8 là một ví dụ của quá trình truy vấn trực tiếp. Nếu xác suất được ước lượng là trên 1 thì chúng ta cần điều chỉnh nó xuống 1. Hàm map sẽ tập hợp tất cả các số lượng và tính toán tất cả sự rối loạn trong các phương thức ghi đè gần nhất. Như đã đề cập xác xuất của mỗi n-gram được tập hợp trong HDFS như đầu ra của reduce cuối cùng.



Hình 3.7: Truy vấn trực tiếp

Chương 4: Các phương pháp đánh giá và thực nghiệm

Trong chương này chúng tôi sẽ trình bày về các phương pháp đánh giá trên sự so sánh về chi phí thời gian và không gian lưu trữ sử dụng các cấu trúc bảng khác nhau. Cũng như độ hỗn loạn mô hình ngôn ngữ cho tập thử nghiệm được đánh giá và so sánh với các công cụ mô hình ngôn ngữ truyền thống.

4.1 Các phương pháp đánh giá

4.1.1 Thời gian và bộ nhớ

Ở đây có hai quá trình chính chúng ta có thể đánh giá, đó là quá trình huấn luyện và quá trình thử nghiệm. Bởi vì có những thử nghiệm trên các cấu trúc bảng khác nhau nên chúng ta sẽ chia quá trình huấn luyện vào 2 bước: Bước đầu là sinh ra số đếm thô và tập hợp thành các tham số trong làm mịn Good-Turing, bước tiếp theo là sinh ra các bảng. Rõ ràng là bước đầu tiên là giống nhau cho tất cả các bảng khác nhau, vì vậy chúng ta sẽ tập chung vào bước thứ 2 để so sánh.

Sự so sánh về chi phí thời gian được dựa trên giá trị trung bình của thời gian chạy chương trình lấy từ nhiều lần chạy để hạn chế độ lệch, độ trễ mạng và các xáo trộn khác có thể ảnh hưởng tới kết quả. Chương trình tính toán thời gian chạy bằng cách so sánh thời gian hệ thống trước và sau công việc MapReduce được thực hiện.

Để thu thập kích thước của các mô hình ngôn ngữ chúng tôi sử dụng các chương trình dòng lệnh cung cấp bởi Hadoop.

4.1.2Sự so sánh độ hỗn loạn thông tin mô hình ngôn ngữ

Cho một mô hình ngôn ngữ, độ hỗn loạn thông tin cho tập thử nghiệm là một đánh giá chung để nhìn thấy cái mô hình nào là tốt. Độ hỗn loạn thông tin có nghĩa là giá trị trung bình được lựa chọn cho mỗi ngram. Nói chung các mô hình tốt hơn đó là độ hỗn tạp thông tin sẽ thấp hơn. Trong một mức của ngram cũng ảnh hưởng đến độ hỗn tạp thông tin cho cùng một mô hình. Cho một kích thước bình thường của tập huấn luyện mô hình ngram ở mức cao hơn sẽ luôn có độ hỗn tạp thông tin thấp hơn. Trong khi đó nhiều tập huấn luyện chúng ta có thì mô hình tốt hơn vì vậy độ hỗn tạp thông tin sẽ trở lên thấp hơn.

Chúng ta cũng có thể so sánh mô hình ngôn ngữ phân tán với công cụ mô hình ngôn ngữ truyền thống như SRILM. SRILM là một gói chức năng phong phú để xây dựng và đánh giá mô hình ngôn ngữ. Các gói phần mềm được viết bằng C++ và bởi vì nó chạy cục bộ trong máy tính duy nhất, tốc độ xử lý nhanh. Các thiếu sót của SRILM là nó sẽ chiếm bộ nhớ và thậm chí tràn bộ nhớ khi xử lý lượng lớn dữ liệu. Chúng ta vẫn có thể so sánh SRILM với mô hình ngôn ngữ phân tán của chúng ta trên cùng một tập huấn luyện. Các phương pháp làm mịn cần phải dùng giống nhau như làm mịn Good-Turing. Các thông số cụ thể khác nhau nhưng phương pháp làm mịn tương tự có thể cho biết mô hình ngôn ngữ phân tán là ổn định hơn với mô hình ngôn ngữ truyền thống.

4.2 Thực nghiệm

Các thực nghiệm được xây dựng trên một môi trường cluster với hai node và một máy chủ. Node được chạy với Hadoop, HDFS và máy chủ thì kiểm soát tất cả. Phần thực nghiệm được thực hiện với quá trình huấn luyện mô hình.

4.2.1 Môi trường chạy thực nghiệm

Các thực nghiệm được chạy trên công cụ mã nguồn mở và được chạy trên máy có cấu hình như sau:

CPU model: Intel ® Core™ i3-2310M CPU@2.10GHz

RAM: 4.00 GB

Hệ điều hành: Ubuntu 15.10, 64 bit

Hadoop: 2.6.0

4.2.2Dữ liệu

Dữ liệu để thực hiện huấn luyện là dữ liệu trên ngôn ngữ tiếng Anh. Dữ liệu được lấy từ website http://www.statmt.org/wmt15/translation-task.html.

Dữ liệu có cấu trúc mỗi câu trên một dòng. Xấp xỉ giữa kích thước dữ liệu và số từ như bảng sau:

Dung lượng	Số từ
790MB	159 triệu

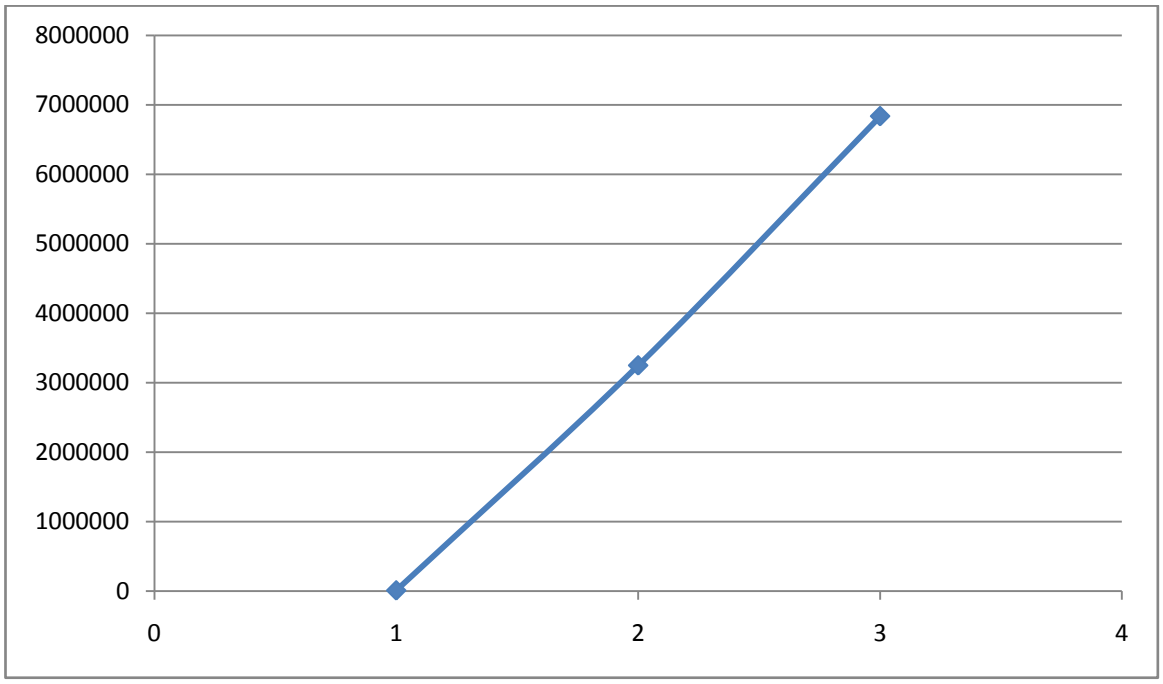
Bảng 4.1: Dữ liệu

4.2.3 Đánh giá thời gian và bộ nhớ cho các ngram

Trong phần thực nghiệm của mình, chúng tôi đã chọn thử nghiệm từ 1-gram tới 3-gram. Một số đếm tỉa bằng 1 được áp dụng cho tất cả các mô hình. Bảng 4.2 sẽ chỉ ra số lượng của 1-gram cho các thứ tựtrên tập huấn luyện. Hình 4.1 sẽ cho biết số lượng 1-gram cho tập huấn luyện.

	Dữ liệu huấn luyện
Số từ	159 triệu
1-gram	411835
2-gram	3248760
3-gram	6834140

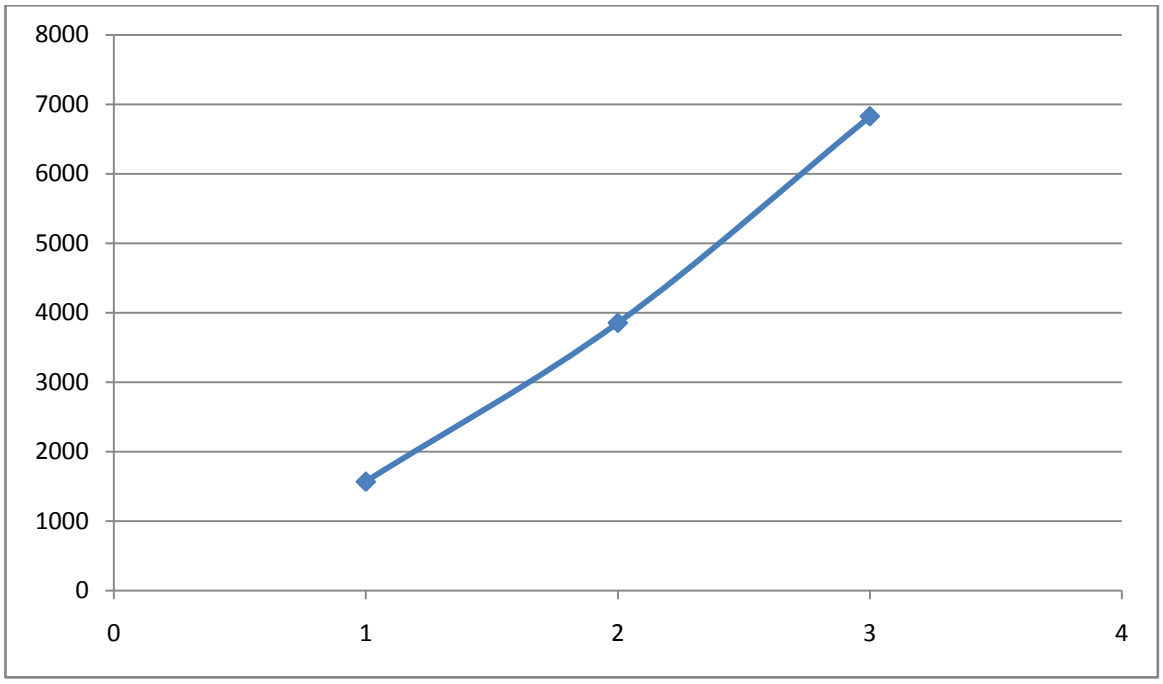
Bảng 4.2: Số lượng n-gram cho các thứ tự khác nhau



Hình 4.1: Số lượng n-gram

	Tập dữ liệu và thời gian chạy
Số từ	159 triệu từ
1-gram	26 phút 10 giây
2-gram	60 phút 14 giây
3-gram	80 phút 45 giây

Bảng 4.3: Thời gian chạy



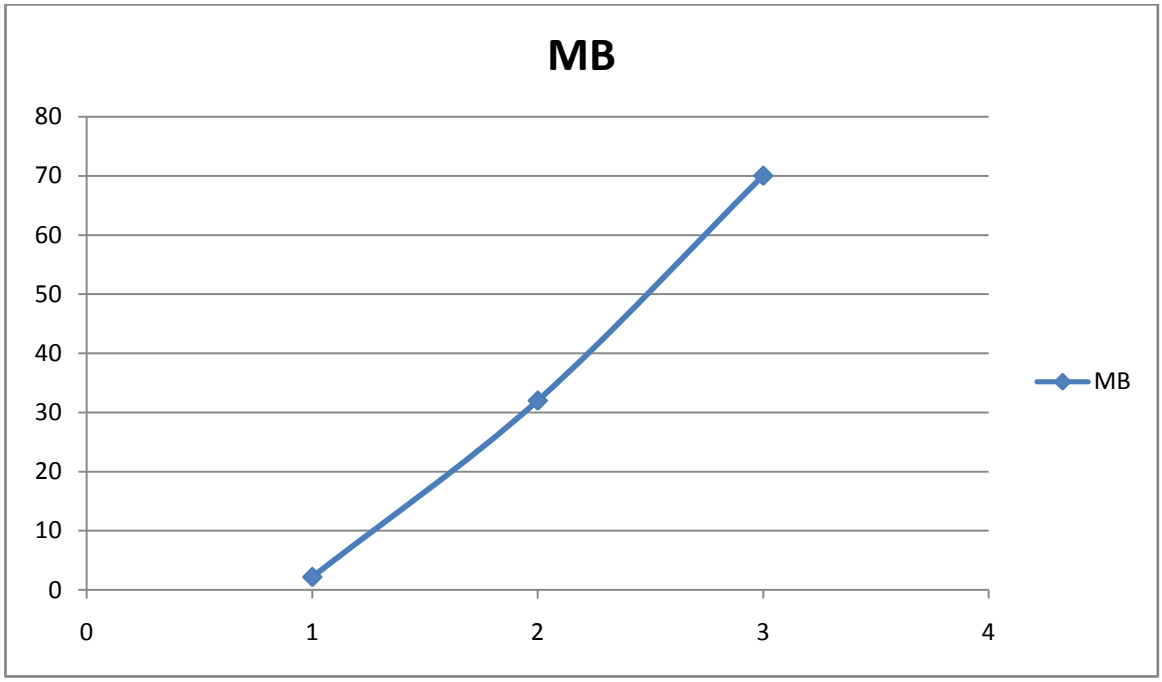
Hình 4.2: Thời gian chạy

Thời gian chạy được đưa ra trong hình 4.2. Chúng ta có thể xem trong bảng 4.3, khi số lượng các từ nhỏ thì thời gian chạy gần như giống nhau, nhưng khi số lượng các từ lớn hơn, thì sự khác nhau giữa các mô hình 1-gram, 2-gram, 3-gram là tăng lên rất nhiều. Một khía cạnh khác ảnh hưởng đến thời gian chạy trong quá trình huấn luyện đó là job MapReduce thứ hai cần phải ước lượng số lần xuất hiện của các từ cho làm mịn Good-Turing. Với job này thì chỉ cần một reduce task dược chạy trong các node và đưa ra một file đầu ra. Khi thứ tự các ngam tăng lên thì số dòng dữ liệu đầu vào cũng tăng lên và sẽ cần nhiều thời gian xử lý hơn

	Dữ liệu huấn luyện, đếm thô và ước lượng
Số từ	159 triệu từ
1-gram	2.2MB
2-gram	32MB
3-gram	70MB

Bảng 4.4: không gian lưu trữ

Bảng 4.4 cho thấy kích thước của các file đếm thô cho mỗi thứ tự n-gram (n=1,2,3) . Hình 4.3 là đồ thị biểu diễn không gian lưu trữ cho các n-gram.



Hình 4.3: Không gian lưu trữ.

4.2.4 So sánh thời gian chạy với SRILM

SRILM (The SRI Language Modeling Toolit) là một công cụ mã nguồn mở để thực nghiệm các bộ dữ liệu trên các mô hình ngôn ngữ chuẩn. Đây là công cụ được sử dụng trong dịch máy thống kê và được sử dụng trong các bài toán về xử lý ngôn ngữ tự nhiên và nhận dạng tiếng nói.

Công cụ SRILM cài đặt các mô hình ngôn ngữ chuẩn, dựa trên thống kê *Ngram* như: Modified Kneser-Ney, Kneser-Ney, Good-Turing, Intepolation, Add-one, Witten-

Bell. Công cụ SRILM dùng để xây dựng mô hình ngôn ngữ N-gram. Công cụ SRILM được cài đặt trên hệ điều hành Ubuntu và câu lệnh chạy như sau:

```
/ngram-count -order 3 -gt1min 3 -gt1max 7 -gt2min 3 -gt2max 7 gt3min 3 -gt3max 7 -text corpus105.txt -lm gt_model.lm
```

Bảng 4.5 là kết quả so sánh khi xây dựng mô hình ngôn ngữ SRILM với kỹ thuật làm mịn good-turing và chương trình viết bằng MapReduce sử dụng cùng kỹ thuật làm mịn Good-Turing.

Dữ liệu	Thời gian chạy SRILM	Thời gian chạy Mapreduce
105MB	100 giây	482 giây
309MB	Không chạy được	20 phút
790	Không chạy được	80 phút

Bảng 4.5: So sánh thời gian chạy

KẾT LUẬN

Luận văn đã trình bày được lý thuyết về mô hình ngôn ngữ: định nghĩa, các phương pháp làm mịn, các phương pháp đánh giá mô hình ngôn ngữ. Tìm hiểu về Hadoop và MapReduce. Phần quan trọng của luận văn là đã tìm hiểu mô hình ngôn ngữ dựa trên Hadoop và MapReduce.

Luận văn đã xây dựng được một ứng dụng bằng MapReduce cho mô hình ngôn ngữ n-gram sử dụng kỹ thuật làm mịn Good-Turing. Luận văn đã chạy thử nghiệm chương trình MapReduce cho quá trình huấn luyện với bộ dữ liệu tiếng anh. Qua các lần chạy thử nghiệm với các bộ dữ liệu khác nhau để so sánh với công cụ SRILM thì đã chứng minh được rằng mô hình ngôn ngữ được xây dựng trên MapReduce có thể thực hiện với bộ dữ liệu lớn và cực lớn trong khi công cụ SRILM thì có thể sẽ không thực hiện được với bộ dữ liệu cực lớn.

Hạn chế của luận văn đó là mới chỉ áp dụng mô hình ngôn ngữ với kỹ thuật làm mịn Good-Turing. Kỹ thuật làm mịn Good-Turing tuy cũng là một kỹ thuật tốt để xây dựng mô hình ngôn ngữ nhưng hiện tại cũng có nhiều kỹ thuật làm mịn khác có thể cho kết quả tốt hơn.

Trong tương lai tôi sẽ tiếp tục nghiên cứu về mô hình ngôn ngữ với MapReduce. Xây dựng thử nghiệm với các kỹ thuật làm mịn khác như kỹ thuật làm mịn Kneser – Ney.

TÀI LIỆU THAM KHẢO

[1] JordanBoyd-Graber.*Language Models.Data-Intensive Information Processing Applications*. 2011.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin. *A Neural Probabilistic Language Model.* Journal of Machine Learning Research 3 (2003) 1137–1155.

[3] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing*. Chapter 4. 2007

[4] Lidstone, G. J. *Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities.* Transactions of the Faculty of Actuaries, 1920, 8, 182–192.

[5] Chen, S. and Goodman, J. *An empirical study of smoothing techniques for language modeling*. Computer Speech & Language, 1999, 13: pages 359-393 (35).

[6] Gale, W.A and Sampson, G. *Good-turing frequency estimation without tears. Journal of Quantitative Linguistics*, 2, 217-237. 1995.

[7] Kneser, R. and Ney, H. *Improved clustering techniques for class-based statistical language modelling.* In EUROSPEECH-93, pp.973-976. 1993.

[8] Irina Sergienya, *Smoothing in Language Modeling*, 2015

[9] Casey McTaggart,*Hadoop/MapReduce,*Object-oriented framework presentation CSCI 5448

[10] Xiaoyang Yu, *Estimating Language Models Using Hadoop and HBase*, 2008

[11] Serge Blazhievsky Nice Systems, *Introduction to Hadoop, MapReduce and HDFS for Big Data Applications*

[12] Klaus Berberich, Srikanta Bedathur, *Computing n-Gram Statistics in MapReduce*, 2013

[13] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, Jeffrey Dean, *Large Language Models in Machine Translation*