ივანე ჯავახიშვილის სახელობის თბილისის
სახელმწიფო უნივერსიტეტი

თორნიკე შავიშვილი

# XML დოკუმენტის პარალელური საძიებო ინდექსის შექმნა

კომპიუტერული მეცნიერება

ნაშრომი შესრულებულია კომპიუტერული მეცნიერებათა მაგისტრის
აკადემიური ხარისხის მოსაპოვებლად

ალექსანდრე გამყრელიძე, თსუ სრული პროფესორი

ლევან კასრაძე, თსუ დოქტორანტი

თბილისი 2013

# ანოტაცია

კვლევის ფარგლებში ჩვენ შევეხეთ საძიებო ინდექსს Inverted index - შებრუნებული ინდექსის მაგალითზე. კომპიუტერულ მეცნიერებაში შებრუნებული ინდექსი ცნობილია როგორც მონაცემთა სტრუქტურა, რომელიც აღწერს ტექსტური ტიპის მონაცემების სტრუქტურას. შებრუნებული ინდექსის დანიშნულებაა მოგვცეს საშუალება განვახორციელოთ სწრაფი ძებნა ტექსტში.

წარმოვადგინეთ დანაწილებული სრული საძიებო ინდექსი და შემოვიღეთ დანაწილების ფუნქცია Q(t,N). დანაწილებული ინდექსი ლოკალიზებულია არა ერთ არამედ რამდენიმე განსხვავებულ ადგილას. ამასთანავე იგი ეფექტურად პასუხობს მომხმარებელთა მოთხოვნებს. ინდექსის დანაწილებაში კრიტიკული ადგილი უკავია Q(t,N)-ს, რომელიც უთანადებს t-სიტყვებს შესაბამის პროცესს.

Q(t,N)-ფუნქცია უნდა აკმაყოფილებდეს რამდენიმე თვისებას: უნდა იყოს დეტერმინისტული, ინექციური და თანაბრად განაწილებული. უკანასკნელი გულისხმობს იმას, რომ N - ცალ პროცესს სიტყვები მაქსიმალურად თანაბარი რაოდენობით უნდა შეუსაბამოს. ეს საჭიროა საუკეთესო საშუალო შედეგის მისაღებად და თავიდან აგვაცილებს კონკრეტული ინდექსების სიტყვებით გადავსებს. ამასთანავე თანაბარი განაწილება იმის გარანტიაა, რომ მიუხედავად მოთხოვნათა რაოდენობისა, პროცესები მაქსიმალურად თანაბრად იქნებიან დაკავებულნი მოთხოვნათა დამუშავებით. წინააღმდეგ შემთხვევაში შეიძლება შეიქმნას სიტუაცია როდესაც ერთი პროცესი ამუშავებს მოთხოვნათა ძირითად ნაწილს და დანარჩენი პროცესები პრაქტიკულად უფუნქციოდ შეიძლება დარჩნენ. არათანაბარი განაწილების დროს და დოკუმენტთა დიდ რაოდენობისას ასევე შესაძლებელია წარმოიქმნას მეხსიერების უკმარისობის პრობლემა კონკრეტული პროცესების ფარგლებში.

ჩვენს მიერ წარმატებით იქნა რეალიზებული Q(t,N) = Hash(t) mod N. ფორმით, წარმატებით გაიარა ტესტირება და დამაკმაყოფილებელი შედეგები აჩვენა: სიტყვათა რაოდენობის გადახრა იდეალური შემთხვევიდან უხეში ზედა შეფასებით არ აჭარბებს Ω(N/2)-ს.

პროექტის ფარგლებში კვლევისა და დემონსტრირებისათვის გამოყენებულია გამოთვლითი და სადემონსტრაციო ინსტრუმენტი Mathematica 9. Mathematica - არის პლატფორმა და დაპროგრამიების ენა რომელიც ორიენტირებულია გამოთვლითი და მათემატიკური პრობლემების გადაწყვეტაზე. მას ავითარებს კომპანია Wolfram Research, დააარსებული 1987 წელს Stephen Wolfram -ის მიერ. მან მოგვცა საშუალება თვალნათელი წარმოდგენა შეგვექმნოდა Q(t,N)- ს ქცევაზე და უფრო ზუსტად შეგვეფასებინა იგი.

ინდექსის რეალიზებისას დავეყრდენით Boost.mpi პარალელურ ბიბლიოთეკას. მას ავითარებს Boost. ამ ბიბლიოთეკის გამოყენება ზრდის პროგრამის ეფექტურობას, ამცირებს მის რეალიზაციაზე დახარჯულ დროს და პროექტში ხარვეზების წარმოქმნის შესაძლებლობას გვაციელბს თავიდან. Boost.mpi წარმოადგენს მრავალპროცესიანი აპლიკაციის რელიზების სტანდარტს. იგი უტლიზაციას უკეთბს C++ ობიექტებს და სერიალიზაციის ტექნოლოგიას. სერიალიზაცია ეს არის პროცესი როდესაც პროგრამული ობიექტი გარდაიქმნება იმდაგვარად, რომ შესაძლებელი გახდეს მისი ფაილში დამახსოვრება, ჩაწერა მეხსიერების მასივში, გადაგზავნა ქსელში იმდაგვარი ფორმატით, რომ ადგილზე შესაძლებელი იყოს ობიექტის ზუსტი ასლის რეკონსტრუქცია. ანუ გვაქვს მოქმედებათა რიგი: სერიალიზაცია - გადაგზავნა - დესერიალიზაცია. სერიალიზაცია იძლევა საშუალებას, რომ პროცესებმა ერთმანეთთან „ისაუბრონ“ ობიექტების ენით. არსებობს სერალიზების რამდენიმე სტანდარტი:

1. Sun Microsystems, External Data Representation(XDR) 1987
2. XML - ფორმატი გამოიყენება Ajax web - აპლიკაციების მიერ
3. JSON - რომელიც ასევე გამიყენება კლიენსა დაერვერს შორის ურთიერთობისას
4. YAML - თავის თავში მოიცავს JSON - ს და შედარებით კომპაქტურია. აქვს მომხმარებლის მიერ განსაზღვრული მონაცემთა ტიპების მხარდაჭერა.

ყურადღება გავამახვილეთ xml დოკუმენტის ტიპზე მისი უნივერსალურობისა და ფართოდ გამოყენებადობის გამო. პლატფორმა Ubuntu 12 - ს ფარგლებში და C++ პროგრამირების ენაზე დაყრდნობით შეიქმნა პროექტი.

კვლევის ფარგლებში მიღწეული შედეგები იძლევა შესაძლებლობას აიგოს მსხვილი საძიებო ინდექსი. კვლევის გაგრძელების შემთხვევაში პროექტს აქვს პოტენციალი გახდეს გამოყენებადი დანაწილებული ინდექსების რეალიზაციის კუთხით. საძიებო სისტემების მნიშვნელოვანი შემედგენელი ნაწილია ე.წ. რანკირების ფუნქცია. მისი დანიშნულებაა

ინდექსში ნაპოვნი დოკუმენტების შეფასება მომხმარებლის მოთხოვნასთან რეველევანტურობის კუთხით. პროექტის ფარგლებში რეალიზაციას ვუკეთებთ შემფასებელ ფუნქციას Okapi BM25. თვითონ ფუნქციის სახელწოდებაა BM25. Okapi - ეს არის სისტემა, რომელიც რეალიზებული იყო London's City University - ში 1980 -1990იან წლებში. ამ სისტემამ პირველად მოახდინა BM25-ს იმპლემენტაცია.

უფრო დაწვრილებით თუ რომელი პარამეტრი რა შინაარსს ატარებს და რა დანიშნულება აქვს, ნაშრომში ცალკე თავში განვიხილავთ.

# შინაარსი

# შესავალი

ინტერნეტი არის კაცობრიობის მიერ დაგროვილი ცოდნისა და გამოცდილების ნაყოფი და პირმშო. მის გამოჩენას წინ უძღვოდა გამოთვლითი მეთოდების სრულყოფისა და განვითარების მრავალსაუკუნოვანი გზა. რაც მეტად სრულყოფილი ხდებოდა გამოთვლითი მეთოდები და გროვდებოდა ცოდნა, მით მეტად წინ მიიწევდა მეცნიერება და მასში შემავალი დარგები: ფიზიკა ქიმია, მათემატიკა... საბოლოოდ კი მათმა განვითარებამ საფუძველი დაუდო ელექტრონულ გამომთვლელ მანქანებს. ინტერნეტი - ეს არის ქსელი. კომპიუტერები ერთიანდებიან ქსელში გარკვეული სახელის ქვეშ და ცვლიან ინფორმაციას გარკვეული წესის - პროტოკოლის მიხედვით. ქსელში არიან „პატარა“ კომპიუტერები - მომხმარებლები, რომლებიც შედიან ინტერნეტში ხსნიან საიტებს ძირითადად დაკავებულნი არიან ინფორმაციის მოძიებით - და სერვერები, სუპერკომპიუტერები რომელთა დანიშნულებაც უმეტესად მომხმარებელთა მომსახურებაა. ჩვენს რეალობაში ინტენეტი - მსოფლიო ქსელი აერთიანებს მილიარდობით მომხმარებელს.

წარმოუდგენლად დიდია მონაცემთა რაოდენობა, რასაც ატარებს ინტენეტი. როგორ გინდა ჩვეულებრივმა მომხმარებელმა მიაგნოს მისთვის სასურველ ინფორმაციას თუ მას არ ექნება სათანადო ინსტრუმენტი? - პასუხი ცალსახაა: ის ამას ვერ მოახერხებს. ინფორმაციის მოძიების პრობლემა ინტერნეტის გამოჩენისთანავე დაისვა, რასაც თავისმხრივ მოჰყვა ვებ - საძიებო სისტემების განვითარება.

მიუხედავად იმისა რომ არსებობს მონაცემთა მრავალი ტიპი, ტექსტური მონაცემები რჩება ინტერნეტში არსებული ინფორმაციის  აღწერისა და წარმოდგენის უნივერსალურ, ყველაზე გავრცელებულ და მოსახერხებელ საშუალებად. თვით ადამიანთა სამეტყველო ენაც წარმოიდგინება ტექსტების საშუალებით.

აქედან გამომდინარე საძიებო სისტემები ფოკუსირებას ახდენენ სწორედ ტექსტურ მონაცმეთა სიმრავლეში ძიებაზე. არაერთი კვლევა მიეძღვნა საძიებო სისტემებს. ტექსტური მონაცემების ძიებისას უმეტესად მიმართავენ ე.წ. საძიებო ინდექსს.

საძიებო ინდექსი (Inverted Index) ინდექსი შეიძლება განხორციელებული იყოს როგორც მონაცემთა ბაზა, ფაილი, განთავსებული იყოს RAM-ში, ნაწილობრივ მყარ დისკზე და ნაწილობრივ ოპერატიულ მეხსიერებაში. მას იყენებენ საძიებო სისტემებში.

პროექტის ფარგლებში რეალიზებულია სრული ინდექსი. ვინაიდან იგი მეტად ფუნქციონალურია ინვერტირებულ ფაილთან შედარებით და იძლევა საშუალებას განხორციელდეს თუნდაც ძიება ფრაზათა მიხედვით.

მოვიყვანოთ ერთერთი ცნობილი მაგალითი:

T[0] = "it is what it is"

T[1] = "what is it"

T[2] = "it is a banana"

უხეშად ვიგულისხმოთ, რომ T[i] განთავსებულია სხვადასხვა ფაილებში. ავაგოთ ინვერტირებული ფაილი:

"a":  {2}

"banana": {2}

"is":  {0, 1, 2}

"it":  {0, 1, 2}

"what":  {0, 1}

ამჯერად იგივე ტექსტზე დაყრდნობით ავაგოთ სრული ინდექსი:

"a":  {(2, 2)}

"banana": {(2, 3)}

"is":  {(0, 1), (0, 4), (1, 1), (2, 1)}

"it":  {(0, 0), (0, 3), (1, 2), (2, 0)}

"what":  {(0, 2), (1, 0)}

მაგალითი გვიჩვენებს, რომ სიტყვა "banana" შეგვხვდა T[2]-ში და იყო რიგით მესამე სიტყვა.

საძიებო ინდექსი წარმოადგენს საძიებო სისტემების ინდექსაციის ალგორითმის ცენტრალურ კომპონენტს. საძიებო სისტემების მიზანია ოპტიმალურ დროში იპოვოს ყველა ის დოკუმენტი, სადაც t სიტყვა გვხვდება. ინდექსის აგება მოითხოვს წინასწარ არსებული დოკუმენტების დამუშავებას. დამუშავების შემდეგ ინდექსი ინახავს თავისთავში დოკუმენტების სიას თითოეულ სიტყვაზე.

ინდექსის შექმნის შემდეგ მოთხოვნის დასამუშავებლად საჭიროა მივაკითხოთ სასურველ სიტყვას.

ინვერტირებულ ინდექსს იყენებენ ასევე ბიომეტრიაში დნმ-ს კვლევისას: მოლეკულის ფრამენტების ძიებისას. მოლეკულა ინდექსის საშუალებით იხლიჩება შემადგენელ ნაწილებად. ადამიანის გენომი შეიცავს დაახლოებით 3 მილიარდ ბაზურ წყვილს და ყველა მისი შემადგენელი ნაწილისთვის საძიებო ინდექსის აგება გეგაბაიტებს საჭიროებს.

საძიებო ინდექსებს აქვთ ერთი თვისება: ისინი მოიხმარენ მეხსიერების დიდ რაოდენობას. მათი მოცულობა საძიებო სივრცის მოცულობის შესადარია და ზოგჯერ აღემატება კიდევაც მას. რა თქმა უნდა პატარა საძიებო სივრცეებში ასეთი სტრუქტურის შემოტანას არ აქვს აზრი. ასეთი სიტუაციებში სრულიად დამაკმაყოფილებელია ძებნის პირდაპირი მეთოდი, როდესაც საძიებო სიტყვა სათითაოდ დარდება საძიებო სივრცის ელემენტებს. თუმცა რეალურ პირობებში ძებნის პირდაპირი მეთოდი გამოუსადეგარია და იძულებულნი ვხდებით მივმართოთ საძიებო ინდექსებს. მეხსიერების დიდი მოთხოვნის დასაკმაყოფილებლად და ამ პრობლემის გადასაწყვეტად მიმართავენ ორ მეთოდს: ზრდიან მეხისერებას და ცდილობენ ძებნა განახორციელონ შემჭიდროვებულ ტექსტებზე.

საძიებო ინდექსი უმეტესად იქმნება საძიებო სისტემის გაშვებისას საწყის ეტაპზე offline რეჟიმში. ფუნქციონირებისათვის მას ესაჭიროება სწრაფი მეხსიერება, ამიტომ იგი განთავსებულია RAM-ში. გარდა ამისა კონკრეტული მანქანის ფარგლებში RAM-ის უსასრულოდ გაზრდაც შეუძლებელია. მეხსიერბის შედარებით დიდ მარაგს შეიცავს HDD - მყარი დისკი, თუმცა ინფორმაციაზე წვდომის მნიშვნელოვნად დაბალი სიჩქარე (100-ჯერ და 1000-ჯერ ნელია) ინდექსებისათვის გამოუყენებლს ხდის.

ამ მეხსიერების და ეფექტურეობის გადასაწყვეტად და დიდი საძიებო ინდექსების განსახორციელებლად კარგი იქნებოდა თუ ერთი საძიებო ინდექსი გაიყოფოდა რამდენიმე ნაწილად და გადაუნაწილდებოდა რამდენიმე სხვადასხვა მანქანას - კომპიუტერთა კლასტერს.

კლასტერი წარმოადგენს კომპიუტერთა გაერთიანებას, როდესაც კომპიუტერები მოქმედებენ ერთმანეთთან შეთანხმებით იმდაგვარად, რომ ისინი აღიქმებიან როგორც ერთი გამომთვლელი სისტემა. იდეა მდგომარეობს იმაში, რომ გაერთიანდეს მრავალი იაფფასიანი კომიუტერის გამოთვლითი შესაძლებლობები ერთ სისტემაში. კლასტერში შემავალი კომპიუტერები ერთმანეთს ძირითადად უკავშირდებიან სწრაფი ლოკალური ქსელის მეშვეობით. ყოველ კვანძზე გაშვებულია საკუთარი ოპერატიული სისტემა. კლასტერში შემავალ კვანძებს ხელმძღვანელობს ე.წ. "clustering middleware", პროგრამული უზრუნველყოფა, რომელიც განთავსებულია თითოეულ კვანძში და აძლევს მომხმარებელს საშუალებას მიმართოს კლასტერს როგორც ერთ გამომთვლელ მანქანას. ერთერთ პირველ კლასტერულ სისტემას წარმოადგენდა „Stone Soupercomputer“ სისტემა. იგი შედგებოდა 133 კვანძისაგან. დეველოპერები ოპერაციულ სისტემად იყენებდნენ Linux-ს, ხოლო კლასტერიზაციისათვის კი მიმართავდნენ Parallel Virtual Machine და MPI-ბიბლიოთეკას. კლასტერთა გამოყენება იძლევა საშუალებას აიგოს შედარებით იაფი სისტემები, რომლებიც ასრულებენ იგივე რაოდენობის გამოთვლებს რასაც ძვირადღირებული სუპერკომპიუტერები და აჭარბებენ კიდევაც მათ.

კლასტერები ლოკალიზებული არიან უმეტესად „ერთი შენობის“ ფარგლებში. მისგან განსხვავებით „Grid“ - ტიპის გაერთიანებებში კომპიუტერები შედარებით ფართო გეოგრაფიულ არეალში არიან განთავსებულნი. ამასთანავე კვანძებთან მიმართებაში „Grid“ მეტად არაერთგვაროვანია. კვანზებიც მეტად თავისუფლად არიან ჩართულნი სისტემის ფუნქციონირებაში, მაშინ როცა კლასტერი ძირითადად აგებულია ერთგვაროვანი კვანძებისაგან.

კლასტერის გამოყენების შემთხვევაში შემთხვევაში მოიხსნებოდა მეხსიერების პრობლემა. ასევე დღის წესრიგში აღარ იქნებოდა ძვირად ღირებული გამოთვლითი ტექნიკის შეძენა და მომსახურება რაც თავისმხრივ დამატებით სახსრებთანაა დაკავშირებული. ვინაიდან კლასტერი შეიძლება შეიქმნას, ელემენტარული პერსონალური კოპმიუტერსებისაგან, რაც საერთო ჯამში ამცირებს სისტემის ღირებულებას და მისი მომსახურების ხარჯებს.

რაც შეეხება „Cloud computing“-ს, ის უფრო აბსტრაქტული ცნებაა. იგი მიზნად ისახავს არსებული სისტემების დადებითი მხარეებისა და შესაძლებლობების გაერთიანებას იმდაგვარად, რომ მომხმარებელს არ დასწჭირდეს ტექნიკურ დეტალებში გარკვევა და ყურადღება გაამახვილოს დასმულ ამოცანაზე. „Cloud computing“-ს შესაძლებელს ხდის ვირტუალიზაცია - მიდგომა რომელიც ახდენს ფიზიკური ინფრასტრუქტურის აბსტრაქციას და ხდის მას ხელმსაწვდომს პროგრამული ურუნველყოფის სახით.

დანაწევრებული საძიებო ინდექსი გულისხმობს საძიებო ინდექსის დანაწილებას და მის ლოკალიზებას არა ერთ, არამედ რამდენიმე ოპერატულ მეხსიერებაში, ასე ვთქვათ რამდენიმე მანქანაზე. ამასთან ნარჩუნდება ძებნის ეფექტურობა ასეთი ინდექსი იძლევა საშუალებას ამასთან მას აქვს უნარი პარალელურად რამდენიმე მოთხოვნა დაამუშაოს და რელაზიაციის კუთხითაც საკმაოდ იოლია.

პროექტის ფარგლებში ჩვენ ვახდენთ განაწილებული საძიებო ინდექსის რეალიზაციას, რისთვისაც გამოვიყენებთ მათემატიკურ ფუნქციას. რეალიზებულ დანაწილებულ ინდექსს ექნება საშუალება მიიღოს და დაამუშაოს მოთხოვნები. ამის შემდეგ შევეხებით ისეთ საკითხებს როგორიცაა საძიებო სისტემში დოკუმენტის დამატება და წაშლა იმდაგვარად, რომ არ გახდეს საჭირო მთლიანი საძიებო ინდექსის ხელახალი აგება.

## ამოცანის დასმა და ძირითადი იდეები

ჩვენ მოვახდენთ დანაწილებული საძიებო ინდექსის რეალიზებას. იდეა მდგომარეობს შემდეგში: ვთქვათ გვაქვს ტექსტური ფაილები. გადავუნაწილოთ ისინი თანაბრად N-ცალ პროცესს, რომელიც გაშვებულია კლასტერზე. პროცესები დაამუშავებენ ფაილებს პარალელურად, საჭირო ინფორმაციას გაცვლიან ერთმანეთს შორის. პროცესის დასრულებისას ჩვენ მივიღებთ საძიებო სტრუქტურას, თუმცა იგი ჯერ კიდევ არ არის დანაწილებული. აუცილებელია აირჩეს მათემატიკური ფუნქცია P(t), რომელიც კონკრეტულ term-ს შეუსაბამებს კონკრეტული პროცესის ინდექსს და ამასთან ამ ფუნქციის გამოყენებით სიტყვები დაახლოებით თანაბრად გადაუნაწილდება პროცესებს. ამ ბიჯზე ჩვენ უკვე მივიღებთ დანაწილებულ ინდექსს და პროგრამა მზადაა უპასუხოს მოთხოვნას.

ამ ეტაპზე საძიებო სტრუქტურა ზემოთ ნახსენები ფორმით პრაქტიკულად სრულადაა რეალიზებული boost.mpi პარალელური ბიბლიოთეკის გამოყენებით. Q(t,N) ფუნქციის

როლში აღებული გვაქვს Q(t,N)=Hash(t) mod N  ფუნქცია, სადაც N პროცესების რაოდენობაა. ფუნქცია შეირჩა შემდეგი მოსაზრებით: რადგანაც term-ის Hash არის uniform - თანაბარი განაწილების, ამიტომ Hash(t) mod N უნდა იყოს თითქმის თანაბად განაწილებული. ჩატარებული ტესტები ამ მოსაზრებას ადასტურებს: სიტყვები მიახლოებით თანაბრად უნაწილდება პროცესებს და შედეგი დამაკმაყოფილებელია.

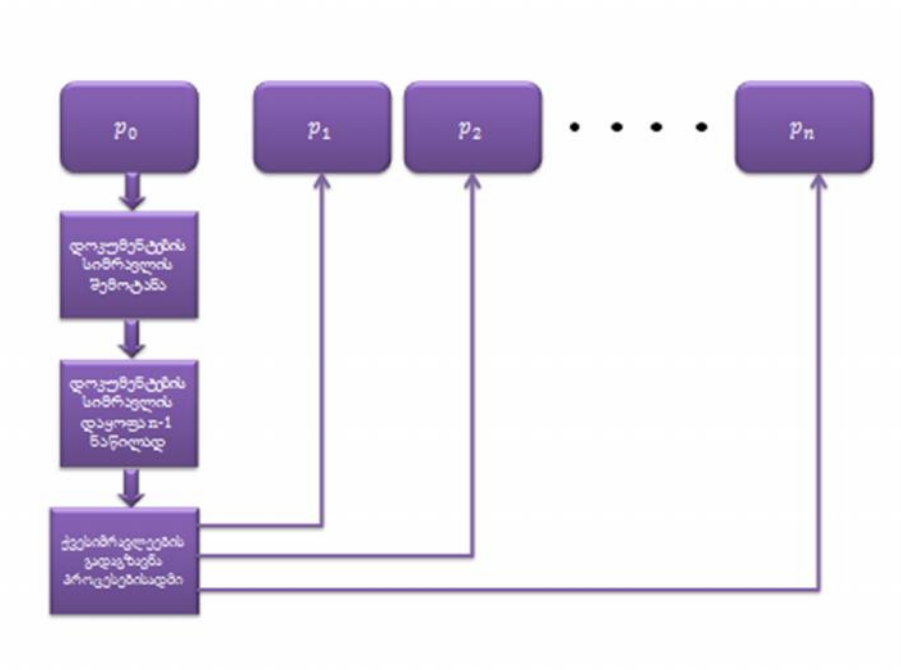
პროგრამა მუშაობს შემდეგნაირად: გაშვებისას ტერმინალიდან მას გადაეცემა პროცესების რაოდენობა და დირექტორია, რომელიც შედგება ფაილებისაგან.

იმისათვის რომ დოკუმენტი ჩაემატოს საძიებო ინდექსში, საჭიროა ჯერ დამუშავდეს სემანტიკურად, რაც გულისხმობს დოკუმენტის სიტყვებად დაშლას და შემდეგ სიტყვები გადაეგზავნოს შესაბამის პროცესებს. ასევე მხედველობაში უნდა იქნეს მიღებული სხვადასხვა მთვლელები და სიტყვების რაოდენობა შესაბამისად უნდა გაიზარდოს. დოკუმენტის წაშლისას ვიქცევით იგივენაირად, ოღონდ მთვლელებს და სიტყვების რაოდენობას ვამცირებთ. ამით ჩვენ ვუზრუნველყოფთ ინდექსის დინამიზმს.

ზემოთ ნახსენები მათემატიკური ფუნქციის არსებობა მნიშვნელოვნად აადვილებს ინდექსის გარდაქმნას იმ შემთხვევაში, როდესაც იცვლება N-პროცესების რაოდენობა. უბრალოდ საჭიროა, რომ განახლებული N-ის პირობებში ყველა სიტყვისთვის ხელახლა შეუსაბამდეს და გადაეგზავნოს პროცესს. ამით ჩვენ თავიდან ვიცილებთ დოკუმენტების ხელახალი დამუშავების აუცილებლობას.

განვიხილოთ ინდექსის სტრუქტურა უფრო დაწვრილებით და დავყოთ ეტაპებად:

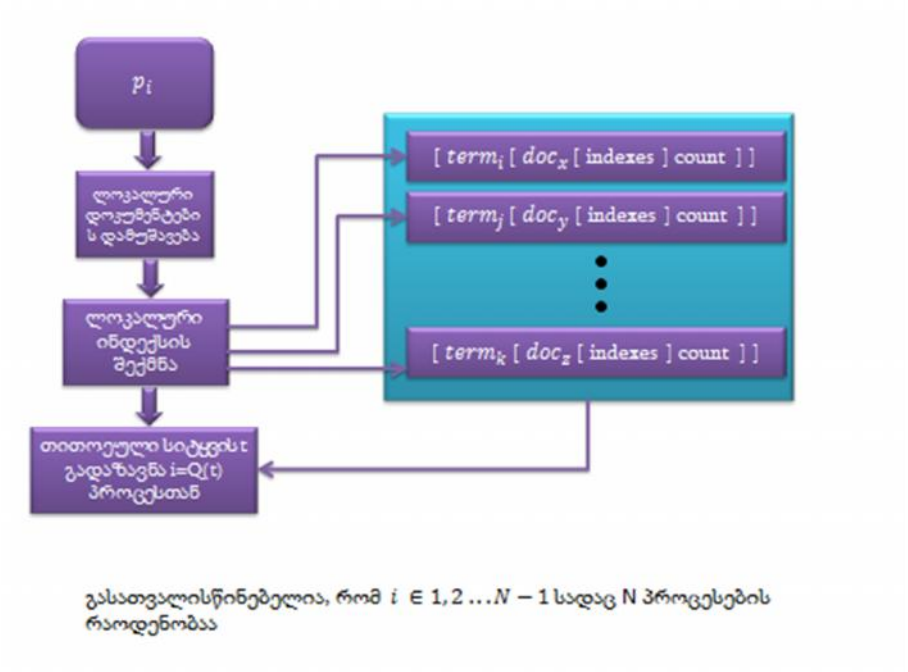
ეტაპი 1 - საძიებო სივრცის დანაწილება




დიაგრამა 1

როგორც დიაგრამიდან ჩანს, დასაწყისში, პირველ ეტაპზე პროგრამა იღებს დოკუმენტთა D სიმრავლეს, რომელზეც უნდა აიგოს საძიებო ინდექსი. დავუშვათ ჯამში გაშვებულია N პროცესი (შემდგომში ვიგულისხმებთ რომ N გაშვებული). 0-ვანი პროცესი, რომელმაც მიიღო D სიმრალვე გაყოფს დაანაწილებს მას (N-1)-ცალ თანაბარ ნაწილად. გაყოფის წესი ელემენტარულია: დავუშვათ $|D| = T$, მაშინ თითოეული ქვესიმრავლის ზომა იქნება $|D_S| = \frac{T}{N-1}$. რაც შეეხება $D_S$ - ქვესიმრავლეთა გადაგზავნას - ყველა პროცესი თავის წილ მონაცემებს იღებენ ერთმანეთის პარალელურად რაც დროის კუთხით მომგებიანია.

ეტაპი 2 - ლოკალური საძიებო ინდექსის აგება




დიაგრამა 2

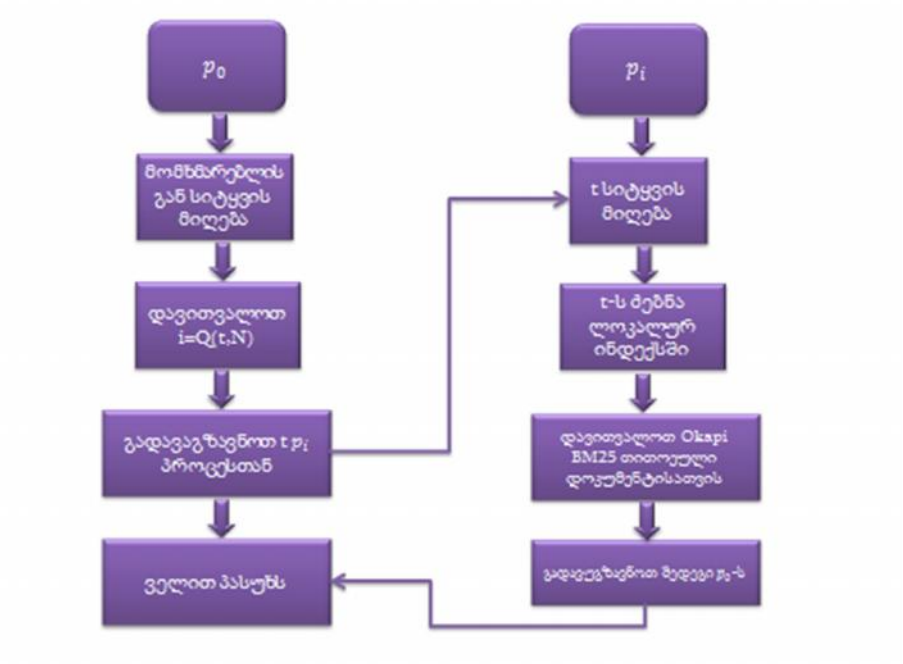
ამ ეტაპზე D უკვე განაწილებულია პროცესებს შორის. თითოეული პროცესი თავის თავში იწყებს ლოკალური საძიებო ინდექსის შექმნას: დაამუშავებს თავის წილ დოკუმენტებს და შექმნის შემდეგი სტრუქტურის მქონე სიას: [term,[doc,[indexes],count]] (აქ [] აღნიშნავს მასივს) რომელიც გვიჩვენებს თუ რომელი term-სიტყვა რომელ დოკუმენტში, რა პოზიციებზე შეგვხვდა და რამდენჯერ. ასევე შესაძლებელია თითოეული დოკუმენტისათვის დაითვალოს ზომა სიტყვებში. $p_i$ პროცესი ყველი term – ს შეუსაბამებს $p_j$-ურ პროცესს, სადაც j = Q(t,N) და გადაუგზავნის მას ინდექსის შესაბამის ნაწილს.

ეტაპი 3 - მზადყოფნა მოთხოვნათა დასამუშავებლად

ამ ეტაპზე $p_j$ პროცესს მიღებული აქვს მისი შესაბამისი term-ები დანარჩენი პროცესებისაგან. $p_j$ უბრალოდ ამოწმებს თუ ერთიდაიგივე term-ი მიიღო რამდენიმე

პროცესიდან, მაშინ მათ გააერთიანებს. გაერთიანება გულისხმობს term-ის შესაბამისი მონაცემების ერთმანეთზე მიბმას. ბოლოს მიღებულ სიტყვებს იგი მოაქცევს სიაში, რის შემდეგაც $p_j$-ს ექნება დანაწილებული ძებნის ინდექსის საკუთარი პორცია და აცხადებს მზადყოფნას დაამუშაოს მოთხოვნები. მოთხოვნათა დამუშავება გულისხმობს შემდეგს: ის მიიღებს term-ს და დააბრუნებს მის შესაბამის მონაცემს: [ term , [ doc, [ indexes ], count ]] -ს.

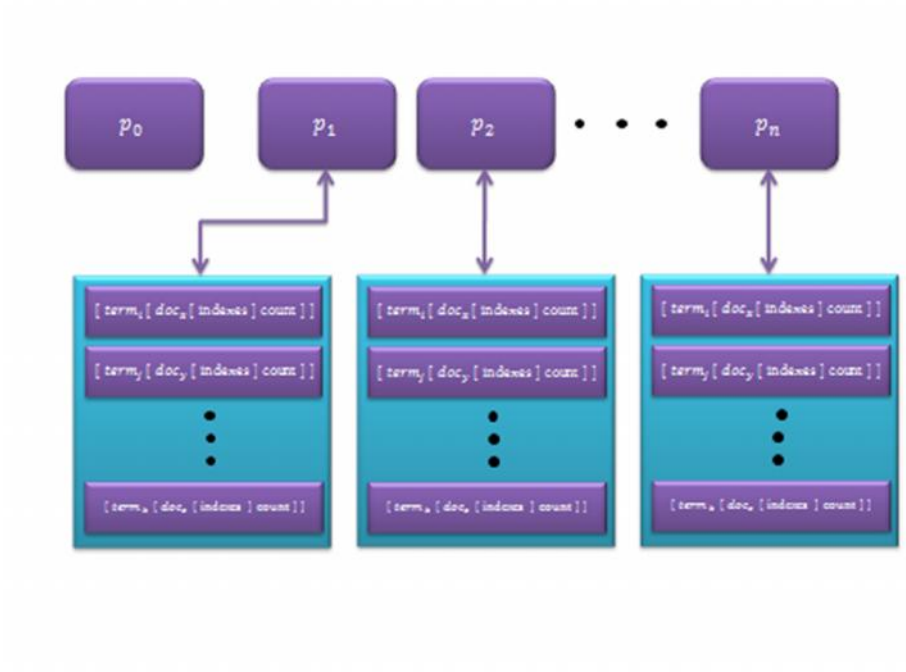
ქვედა დიაგრამაზე გრაფიკულად წარმოდგენილია თუ როგორ ემსახურება პროგრამა ერთსიტყვიან მოთხოვნებს:




მოთხოვნის დამუშვების დიაგრამა 1

ძებნის მოცემული გზით განხორციელებას აქვს რამდენიმე უპირატსობა: ჯერ ერთი შესაძლებელი ხდება რამდენიმე მოთხოვნა დამუშავდეს პარალელურად. ამასთან თითოეული სიტყვის ძებნა მიმდინარეობს არა მთლიან ინდექსში, ზომით $|D|$, არამედ მის კონკრეტულ ქვესიმრავლეში $|D_S| = \frac{|D|}{N-1}$, რაც ამცირებს ძებნის სივრცეს და შესაბამისად ზრდის სისტემის ეფექტურობას.

ამ ეტაპების გავლის შემდეგ ინდექსი მიიღებს შემდეგ სახეს




დიაგრამა 3

როგორც ვხედავთ 0-ოვანი პროცესი არ იღებს ძებნაში მონაწილეობას. მისი დანიშნულება არის მოთხოვნათა განაწილება დანარჩენ პროცესებს შორის.

Q(t,N) დანაწილების ფუნქცია

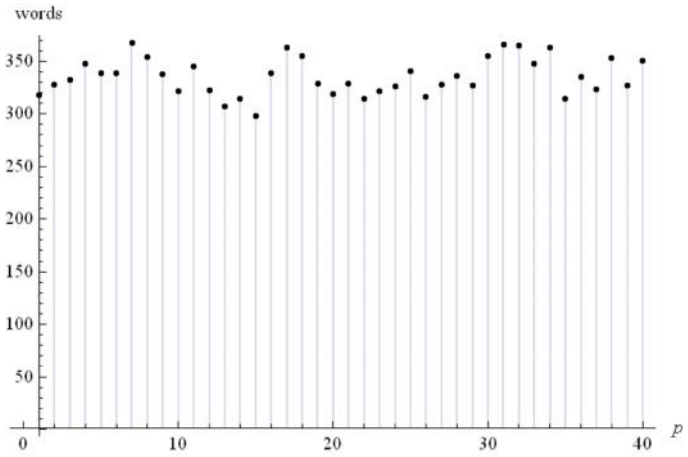
ჩვენ ვნახეთ, რომ საძიებო ინდექსის განაწილება Q(t,N) ფუნქციას. მისი დანიშნულებაა t – term-ი ცალსახად შეუსაბამოს ერთ კონკრეტულ პროცესს. პროცესები არის გადანორმრილი და მათ რაოდენობა წინასწარ ცნობილია, ასევე ცნობილია სიტყვებიც. Q(t,N)-ი უნდა აკმაყოფილებდეს შემდეგ პირობებს:

1. უნდა იყოს დეტერმინისტული
2. უნდა იყოს ინექციური
3. მარტივად გამოთვლადი
4. მოცემულ სიტყვათა სიმრავლეზე მან სიმრალვე მასიმალურად თანაბარ (N-1)-ცალ ნაწილად უნდა გაყოს

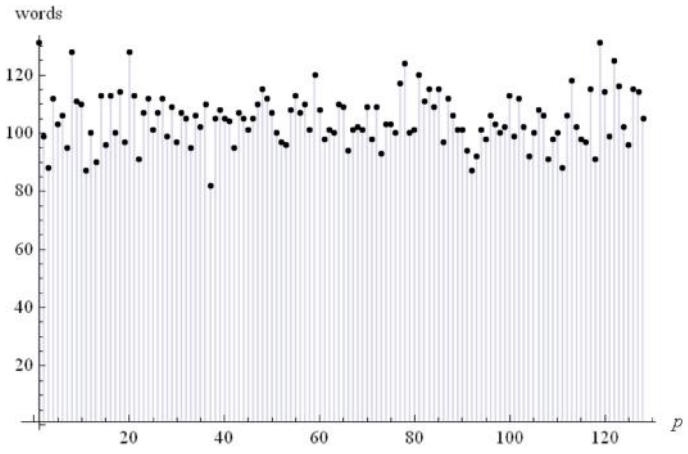
ვინაიდან ჩვენი მიზანია ფუნქცია იყოს თანაბრად განაწილებული, მოვიქცეთ შემდეგნაირად: დავთვალოთ term-ის hash-ი, ამის შემდეგ კი მიღებულ რიცხვს დავითვლით mod N.

განვიხილოთ ფუნქცია Q(t,N)=Hash(t) mod N იგი აკმაყოფილებს ჩვენს მიერ წამოყენებულ პირობებს: არის დეტერმინისტული, არის ინექციური და მარტივად გამოთვლადი. რაც შეეხება იგი დამაკმაყოფილებლად პასუხობს მეოთხე პირობას, რის დემონსტრირებასაც მოვახდენთ Mathematica 9 - საშუალებით.

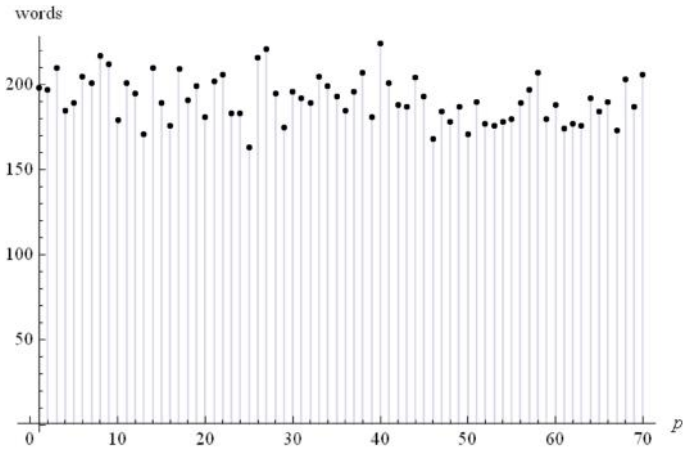
გრაფიკულად წარმოვადგენთ ფუნქციის მუშაობის შედეგებს სიტყვათა კონკრეტულ სიმრავლზე. როგორც დიაგრამებიდან ჩანს სიტყვათა რაოდენობა პროცესებს შორის მიახლოებით ერთნაირია. ტესტირებებმა აჩვენა, რომ Q(t,N)-ს მოცემული წესით იმპლემენაციისას სიტყვათა რაოდენობა იდეალური შემთხვევისაგან განსხვავდება არაუმეტეს Ω(N/2)-ით. პროექტის ფარგლებში ფუნქციამ წარმატებით გაიარა ტესტირება და დამაკმაყოფილებელი შედეგები აჩვენა:



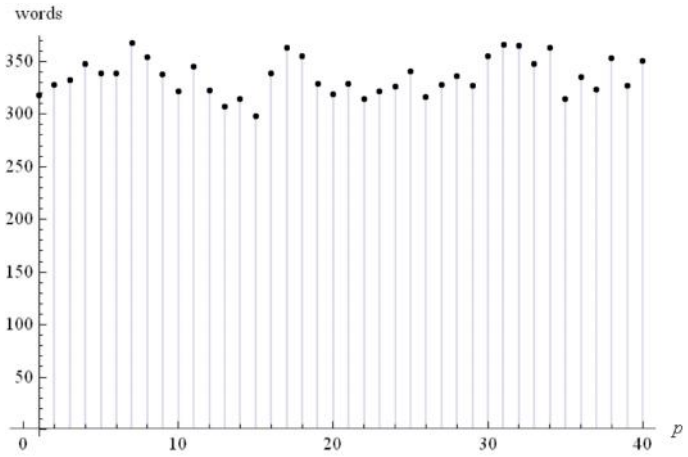

დიაგრამა Q(t,N) 1




დიაგრამა Q(t,N) 2




დიაგრამა Q(t,N) 3




დიაგრამა Q(t,N) 4

Okapi BM25 შემფასებელი ფუნქცია

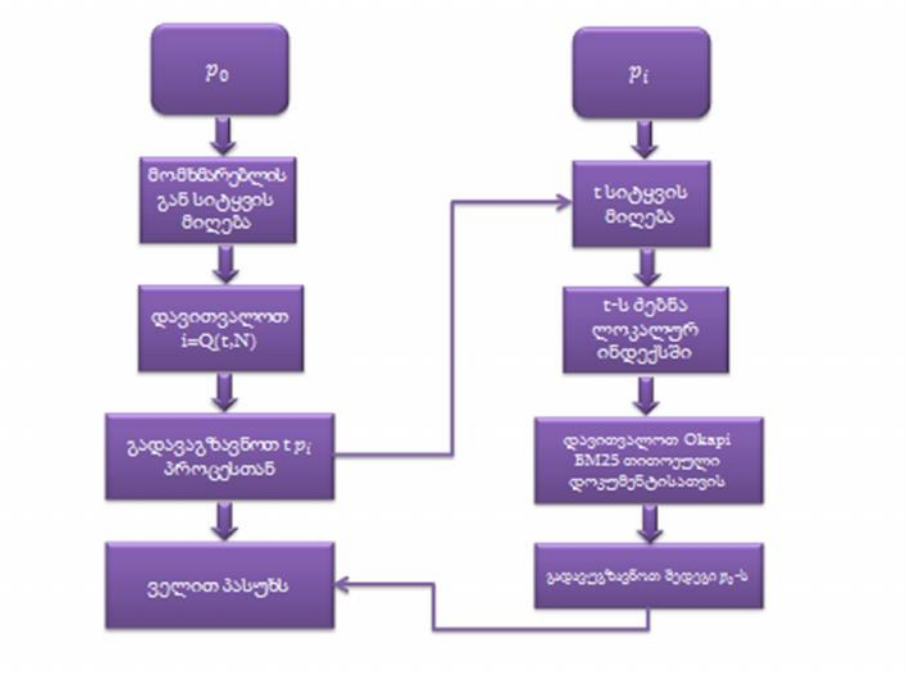
როგორც უკვე აღვნიშნეთ საძიებო სისტემის მნიშვნელობანი შემედგენელი ნაწილია ე.წ. რანგირების ფუნქცია. მისი დანიშნულებაა ინდექსში ნაპოვნი დოკუმენტების შეფასება მომხმარებლის მოთხოვნასთან რეველევანტურობის კუთხით. პროექტის ფარგლებში რეალიზაციას ვუკეთებთ შემფასებელ ფუნქციას Okapi BM25. თვითონ ფუნქციის სახელწოდებაა BM25. Okapi - ეს არის სისტემა, რომელიც რეალიზებული იყო London's City University - ში 1980 -1990იან წლებში. ამ სისტემამ პირველად მოახდინა BM25-ს იმპლემენტაცია.

სისტემაში Okapi BM25-ს რეალიზიაცია მოხდა შემდეგი ფორმით, ვთქვათ მოცემულია მოთხოვნა Q = $q_1, q_2, \ldots q_n$ ,სადაც D დოკუმენტის BM25 შეფასება დაითვლება შემდეგნაირად:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D) \cdot (k_1+1)}{f(q_i,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

სადაც $f(q_i, D)$-არის $q_i$-ის სიხშირე D დოკუმენში, $|D|$ - დოკუმენტის ზომა სიტყვებში, $avgdl$-საძიებო სივრცეში დოკუმენტის საშუალო ზომა სიტყვებში, $k_1 \in [1.2, 2.0]$ და b=0.75 თავისუფალი პარამეტრებია რომლებსაც ცვლიან შეფასებების ოპტიმიზაციისათვის. N-დოკუმენტთა რაოდენობა, ხოლო $n(q_i)$-დოკუმენტების რაოდენობა, რომლებიც შეიცავენ $q_i$-ს. ამ ფუნქციის გათვალისწინებით ზემოთ მოყვანილი დიაგრამები გამოიყურებიან შემდეგნაირად:




შეფასებები იძლევა საშუალებას მოხდეს შედეგების მარტივი სორტირება და შემდეგ წარდგენა მომხმარებლისადმი.

## პროგრამის განხორციელება

N-ცალი პროცესის გაშვების შემდეგ პროგრამა 0-ოვან პროცესში იღებს დოკუმენტების მისამართს. შემდეგი ფუნქცია დირექტორიიდან რეკურსიულად ამოკრეფს ყველა xml ფაილის სრულ სახელს:

```
void parseDirectory(string dir, vector<string> &result){
      string f_ext(".xml");
      const path startDirectory(dir);
    // First check if the start path exists
    if (!exists(startDirectory) || !is_directory(startDirectory)){ return;
}

    recursive_directory_iterator it(startDirectory); // Directory iterator
at the start of the directory
    recursive_directory_iterator end;                // Directory iterator
by default at the end

    // Iterate all entries in the directory and sub directories
    while (it!=end)
    {
         if (!is_directory(it->status())){
```

```
            if(f_ext.compare(it->path().extension().string())==0){
                  result.push_back(it->path().string());
            }
         }

         // When a symbolic link, don't iterate it. Can cause infinite
loop.
         if (is_symlink(it->status())) it.no_push();

         // Next directory entry
         it++;
    }
}
```

მიღებული შედეგი იყოფა N-1 თანაბარ ნაწილად:

```
void splitVector(vector<string> &files, int n, vector<vector<string>>
&subvectors){
      int max_size;
      int part_size;
      if (files.size() <= n) {
            part_size = 1;
            max_size = files.size();
      } else {
            part_size = files.size() / n;
            max_size = n;
      }
      for(int i=0; i<max_size; i++){//cout<<"t "<<max_size<<"
"<<part_size<<endl;
            vector<string> item(
                        files.begin()+i*part_size,
files.begin()+(i+1)*part_size
                        );
            subvectors.push_back(item);
      }
      if(files.size()%n !=0 && files.size() > n){
            vector<string>
item(files.rbegin(),files.rbegin()+files.size()%n);
            (subvectors.end()-1)->insert((subvectors.end()-1)-
>end(),item.begin(),item.end());
      }
      subvectors.resize(n);
}
```

ამის შემდეგ Boost.mpi ბიბლიოთეკის გამოყენებით ნაწილები გადაიგზავნება პროცესებში:

`world.isend(i + 1, 1, subvectors.at(i));` სადაც `i` - პროცესის ინდექსია. პრცესები მიიღებენ თავის წილ დირექტორიას და დამუშავებენ ფაილებს, რის შედეგედაც აიგება ლოკალური საძიებო ინდექსი.

ფაილებს ამუშავებს შემდეგი ფუნქციათა ჯგუფი:

```
void parseFile(string file, map<string,int> &file_dict, int &size_in_words){
	pugi::xml_document doc;
	pugi::xml_parse_result result = doc.load_file(file.c_str());
	if (result.status == pugi::xml_parse_status::status_ok) {
		auto root = doc.document_element();
		recursiveInsert(root,file_dict);
		size_in_words=0;
		for(auto &item:file_dict){
			size_in_words+=item.second;
		}
	}else{
		size_in_words=-1;
		file_dict.clear();
	}

}

void recursiveInsert(pugi::xml_node &root,
		map<string, int> &file_dict){
	string root_name = root.name();
	string root_value = root.value();
	insertWord(root_name,file_dict);
	insertWord(root_value,file_dict);

	auto children = root.children();
	for (auto &node : children) {
		recursiveInsert(node,file_dict);
	}
}

void insertWord(string &word, map<string,int> &map){
	stringstream trimmer;
	if (!word.empty()) {
		trimmer << word;
		word.clear();
	}
	while(trimmer >> word){
		if (!word.empty()) {
				map[word]+=1;
		}
	}
	}
```

ლოკალური ინდექსში ყოველი სიტყვა უსაბამდება პროცესს შემდეგი ფუნქციის გამოყენებით:

```
int correspondsToProcess(string word, int total_proc_count){
       boost::hash<string> string_hash;
     int proc = ((unsigned)string_hash(word))%(total_proc_count-1)+1;
     return proc;
}
```

შედეგზე ერთის მიმატება საჭიროა იმისათვის პროცესის ინდექსი დაიწყოს არა 0-იდან, არამედ 1-დან.

თითოეული სიტყვის შესაბამისი მონაცემები გადაეგზავნება შესაბამის პროცესს `world.isend(i, 2, word_list);` ფუნქციის საშუალებით, რის შემდეგაც პროგრამა აცადებს მზადყოფნას ერთსიტყვიანი მოთხოვნების დამუშავებაზე: 0-ოვანი პროცესი მიიღებს სიტყვას, შეუსაბამებს მას პროცესს და გადაუგზავნის. საკუთრივ პროცესი მიიღებს სიტყვას და ლოკალურ ინდექსიდან დააბრუნებს სიტყვის შესაბამის სიას - ან ცარიელ სიას იმ შემთხვევაში, თუ სიტყვა არ მოიძებნა

როგორც ვთქვით, პროგრამა თითოეული დოკუმენტისათვის ითვლის Okapi BM25 ქულას. მისი ერთერთი პარამეტრია დოკუმენტთა საშუალო ზომა სიტყვებში. ვინაიდან ინდექსი დანაწილაბულია, ყველა „ქვეინდექსი“ ითვლის სიტყვათა ჯამს დოკუმენტევისათვის. ამის შემდეგ ჩვენ ვიყენებთ ფუნქციას `reduce(world, local_avgdl, avgdl, summarize<float>(), 0);` ეს ფუნქცია პარამეტრად ფუნქციის ობიექტს - `summarize<float>()`. ფუნქციის ობიექტი ნიშნავს თუ რა ოპერაცია უნდა განახორციელოს მან პარამეტრად გადაცემულ ცვლადზე. შემდეგ ცვლადები, რომელზზეს ამ ოპერაციას ვასრლებთ, ცვლადი, რომელშიც უნდა დააბრუნოს შედეგი და პროცესის ინდექსი სადაც აბრუნებს შედეგს. მისი გამოყენებით მისი გამოყენებით ჩვენ ავჯამავთ ცვლადებს თითოეული პროცესიდან და 0-ოვან პროცესში დავითვლით დოკუმენტთა საშუალო ზომას.

# დასკვნა

მოცემულ ნაშრომში ჩვენ მიმოვიხილეთ საძიებო ინდექსები. შევეხეთ ისეთ საკითხებს, როგორიცაა გამოყენებული მეხსიერება, დინამიზმი. შემოვიტანეთ საძიებო ინდექსის დანაწილების მეთოდი პროცესებს შორის მათემატიკური ფუნქციის გამოყენებით და ასევე შევძელით აგვეგო დანაწილებული საძიებო ინდექსის მუშა მაგალითი. რეალიზება გავუკეთეთ დოკუმენტის შემფასებელ ფუნქციას. გავეცანით სერალიზაციის ტექნოლოგიას და გამოვიყენეთ იგი პროცესებს შორის ობიექტების ენაზე „სასაუბროდ“. გავეცანით და წარმატებით გამოვიყენეთ პლატფორმა Mathematica ფუნქციათა მუშობის გრაფიკულად წარმოდგენისათვის.

სერიალიზაცია გამოიყენება ყოველთვის, როდესაც ერთ პროცესს სურს გარკვეული მონაცემები გადაუგზავნოს სხვა პროცესს, ეს იქნება ძებნის შედეგი, დანაწილებული ინდექსის ნაწილი მისი აგებისას, დანაწილებული დოკუმენტთა სივრცე და ა.შ.

საძიებო ინდექსის მნიშვნელობა და როლი მნიშვნელოვნად გაიზრდება დროის სვლასთან ერთად, თუ ადამიანთა ცივილიზაციამ განვითარების არსებული ვექტორი შეინარჩუნა. შესაბამისად საჭიროა გადაიჭრას ნაწილობრივ ღიად არსებული საკითხები, რომელთა შორისაა მოხმარებული მეხსიერება და ინდექსის განახლება.

# გამოყენებული ლიტერატურა


1. ვიკიპედია, Okapi_BM25: http://en.wikipedia.org/wiki/Okapi_BM25
2. ვიკიპედია, Inverted_index: http://en.wikipedia.org/wiki/Inverted_index
3. Boost ბიბლიოთეკა: http://www.boost.org/doc/libs/1_49_0/doc/html/mpi/tutorial.html
4. Boost.MPI ბიბლიოთეკა: http://www.boost.org/doc/libs/1_39_0/doc/html/mpi.html
5. Open.MPI ბიბლიოთეკა: http://www.open-mpi.org/doc/
6. Mpich ბიბლიოთეკა: http://www.mpich.org/documentation/guides/
7. Inverted Files for Text Search Engines: JUSTIN ZOBEL-RMIT University, Australia ALISTAIR MOFFAT-The University of Melbourne, Australia
8. Compressed Full-Text Indexes: Gonzallo Navaro, University of chile, Veli Makinen, University of Helsinki

დანართი

# Okapi BM25

From Wikipedia, the free encyclopedia

In information retrieval, **Okapi BM25** is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. It is based on the probabilistic retrieval framework developed in the 1970s and 1980s by Stephen E. Robertson, Karen Spärck Jones, and others.

The name of the actual ranking function is BM25. To set the right context, however, it usually referred to as "Okapi BM25", since the Okapi information retrieval system, implemented at London's City University in the 1980s and 1990s, was the first system to implement this function.

BM25, and its newer variants, e.g. BM25F (a version of BM25 that can take document structure and anchor text into account), represent state-of-the-art TF-IDF-like retrieval functions used in document retrieval, such as Web search.

## The ranking function

BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity). It is not a single function, but actually a whole family of scoring functions, with slightly different components and parameters. One of the most prominent instantiations of the function is as follows.

Given a query $Q$, containing keywords $q_1, \cdots, q_n$, the BM25 score of a document $D$ is:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

where $f(q_i, D)$ is $q_i$'s term frequency in the document $D$, $|D|$ is the length of the document $D$ in words, and $avgdl$ is the average document length in the text collection from which documents are drawn. $k_1$ and $b$ are free parameters, usually chosen, in absence of an advanced optimization, as $k_1 \in [1.2, 2.0]$ and $b = 0.75$ $\text{IDF}(q_i)$ is the IDF (inverse document frequency) weight of the query term $q_i$. It is usually computed as:

$$\text{IDF}(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5},$$

where $N$ is the total number of documents in the collection, and $n(q_i)$ is the number of documents containing $q_i$.

There are several interpretations for IDF and slight variations on its formula. In the original BM25 derivation, the IDF component is derived from the Binary Independence Model.

Please note that the above formula for IDF shows potentially major drawbacks when using it for terms appearing in more than half of the corpus documents. These terms' IDF is negative, so for any two almost-identical documents, one which contains the term and one which does not contain it, the latter will possibly get a larger score. This means that terms appearing in more than half of the corpus will provide negative contributions to the final document score. This is often an undesirable behavior, so many real-world applications would deal with this IDF formula in a different way:

- Each summand can be given a floor of 0, to trim out common terms;
- The IDF function can be given a floor of a constant $\epsilon$, to avoid common terms being ignored at all;
- The IDF function can be replaced with a similarly shaped one which is non-negative, or strictly positive to avoid terms being ignored at all.

IDF Information Theoretic Interpretation

---

Here is an interpretation from information theory. Suppose a query term $q$ appears in $n(q)$ documents. Then a randomly picked document $D$ will contain the term with probability $\frac{n(q)}{N}$ (where $N$ is again the cardinality of the set of documents in the collection). Therefore, the information content of the message "$D$ contains $q$" is:

$$-\log \frac{n(q)}{N} = \log \frac{N}{n(q)}.$$

Now suppose we have two query terms $q_1$ and $q_2$. If the two terms occur in documents entirely independently of each other, then the probability of seeing both $q_1$ and $q_2$ in a randomly picked document $D$ is:

$$\frac{n(q_1)}{N} \cdot \frac{n(q_2)}{N},$$

and the information content of such an event is:

$$\sum_{i=1}^{2} \log \frac{N}{n(q_i)}.$$

With a small variation, this is exactly what is expressed by the IDF component of BM25.

Modifications

---

- At the extreme values of the coefficient $b$ BM25 turns into ranking functions known as **BM11** (for $b = 1$) and **BM15** (for $b = 0$)
- **BM25** is a modification of BM25 in which the document is considered to be composed from several fields (such as headlines, main text, anchor text) with possibly different degrees of importance.

# Inverted index

From Wikipedia, the free encyclopedia

In computer science, an **inverted index** (also referred to as **postings file** or **inverted file**) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrievalsystems, used on a large scale for example in search engines. Several significant general-purpose mainframe-based database management systems have used inverted list architectures, including ADABAS, DATACOM/DB, and Model 204.

There are two main variants of inverted indexes: A **record level inverted index** (or **inverted file index** or just **inverted file**) contains a list of references to documents for each word. A **word level inverted index** (or **full inverted index** or **inverted list**) additionally contains the positions of each word within a document.The latter form offers more functionality (like phrase searches), but needs more time and space to be created.

Example

Given the texts

```
T[0] = "it is what it is"
T[1] = "what is it"
T[2] = "it is a banana"
```

we have the following inverted file index (where the integers in the set notation brackets refer to the indexes (or keys) of the text symbols, T[0], T[1] etc.):

```
"a":      {2}
```

"banana": {2}

"is": {0, 1, 2}

"it": {0, 1, 2}

"what": {0, 1}

A term search for the terms "what", "is" and "it" would give the set $\{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}$.

With the same texts, we get the following full inverted index, where the pairs are document numbers and local word numbers. Like the document numbers, local word numbers also begin with zero. So, "banana": {(2, 3)} means the word "banana" is in the third document (T[2]), and it is the fourth word in that document (position 3).

"a": {(2, 2)}

"banana": {(2, 3)}

"is": {(0, 1), (0, 4), **(1, 1)**, (2, 1)}

"it": {(0, 0), (0, 3), **(1, 2)**, (2, 0)}

"what": {(0, 2), **(1, 0)**}

If we run a phrase search for "what is it" we get hits for all the words in both document 0 and 1. But the terms occur consecutively only in document 1.

Applications

The inverted index data structure is a central component of a typical search engine indexing algorithm. A goal of a search engine implementation is to optimize the speed of the query: find the documents where word X occurs. Once a forward index is developed, which stores lists of words per document, it is next inverted to develop an inverted index. Querying the forward index would require sequential iteration through each document and to each word to verify a matching document. The time, memory, and processing resources to perform such a query are not always technically realistic. Instead of listing

the words per document in the forward index, the inverted index data structure is developed which lists the documents per word.

With the inverted index created, the query can now be resolved by jumping to the word id (via random access) in the inverted index.

In pre-computer times, concordances to important books were manually assembled. These were effectively inverted indexes with a small amount of accompanying commentary that required a tremendous amount of effort to produce.

In bioinformatics, inverted indexes are very important in the sequence assembly of short fragments of sequenced DNA. One way to find the source of a fragment is to search for it against a reference DNA sequence. A small number of mismatches (due to differences between the sequenced DNA and reference DNA, or errors) can be accounted for by dividing the fragment into smaller fragments—at least one subfragment is likely to match the reference DNA sequence. The matching requires constructing an inverted index of all substrings of a certain length from the reference DNA sequence. Since the human DNA contains more than 3 billion base pairs, and we need to store a DNA substring for every index, and a 32-bit integer for index itself, the storage requirement for such an inverted index would probably be in the tens of gigabytes.

# Grid computing

**Grid computing** is the federation of computer resources from multiple locations to reach a common goal. The **grid** can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed.Although a single grid can be dedicated to a particular application, commonly a grid is used for a variety of purposes. Grids are often constructed with general-purpose grid middleware software libraries.

Grid size varies a considerable amount. Grids are a form of distributed computing whereby a **"super virtual computer"** is composed of many networked loosely coupledcomputers acting together to perform large tasks. For certain applications, "distributed" or "grid" computing, can be seen as a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface, such as Ethernet. This is in contrast to the traditional notion of a supercomputer, which has many processors connected by a local high-speed computer bus.

## Overview

A grid computer is multiple number of same class of computers clustered together. A grid computer is connected through a super fast network and share the devices like disk drives, mass storage, printers and RAM. Grid Computing is a cost efficient solution with respect to Super Computing. Operating system has capability of parallelism

Grid computing combines computers from multiple administrative domains to reach **a common goal to solve a single task**, and may then disappear just as quickly.

One of the main strategies of grid computing is to use middleware to divide and apportion pieces of a program among several computers, sometimes up to many thousands. Grid computing involves computation in a distributed fashion, which may also involve the aggregation of large-scale clusters.

The size of a grid may vary from small—confined to a network of computer workstations within a corporation, for example—to large, public collaborations across many companies and networks. "The notion of a confined grid may also be known as an intra-nodes cooperation whilst the notion of a larger, wider grid may thus refer to an inter-nodes cooperation"

Grids are a form of distributed computing whereby a "super virtual computer" is composed of many networked loosely coupled computers acting together to perform very large tasks. This

technology has been applied to computationally intensive scientific, mathematical, and academic problems through volunteer computing, and it is used in commercial enterprises for such diverse applications as drug discovery, economic forecasting, seismic analysis, and back office data processing in support for e-commerce and Web services.

Coordinating applications on Grids can be a complex task, especially when coordinating the flow of information across distributed computing resources. Grid workflow systems have been developed as a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or a workflow, in the Grid context.

## Comparison of grids and conventional supercomputers

"Distributed" or "grid" computing in general is a special type of parallel computing that relies on complete computers (with onboard CPUs, storage, power supplies, network interfaces, etc.) connected to a network (private, public or the Internet) by a conventional network interface producing commodity hardware, compared to the lower efficiency of designing and constructing a small number of custom supercomputers. The primary performance disadvantage is that the various processors and local storage areas do not have high-speed connections. This arrangement is thus well-suited to applications in which multiple parallel computations can take place independently, without the need to communicate intermediate results between processors.The high-end scalability of geographically dispersed grids is generally favorable, due to the low need for connectivity between nodes relative to the capacity of the public Internet.[*citation needed*]

There are also some differences in programming and deployment. It can be costly and difficult to write programs that can run in the environment of a supercomputer, which may have a custom operating system, or require the program to address concurrency issues. If a problem can be adequately parallelized, a "thin" layer of "grid" infrastructure can allow conventional, standalone programs, given a different part of the same problem, to run on multiple machines. This makes it possible to write and debug on a single conventional machine, and eliminates complications due to multiple instances of the same program running in the same shared memory and storage space at the same time.

## Design considerations and variations

One feature of distributed grids is that they can be formed from computing resources belonging to multiple individuals or organizations (known as multiple administrative domains). This can facilitate commercial transactions, as in utility computing, or make it easier to assemble volunteer computing networks.[*citation needed*]

One disadvantage of this feature is that the computers which are actually performing the calculations might not be entirely trustworthy. The designers of the system must thus introduce measures to prevent malfunctions or malicious participants from producing false, misleading, or erroneous results, and from using the system as an attack vector. This often involves assigning work randomly to different nodes (presumably with different owners) and checking that at least two different nodes report the same answer for a given work unit. Discrepancies would identify malfunctioning and malicious nodes.

Due to the lack of central control over the hardware, there is no way to guarantee that nodes will not drop out of the network at random times. Some nodes (like laptops or dialupInternet customers) may also be available for computation but not network communications for unpredictable periods. These variations can be accommodated by assigning large work units (thus reducing the need for continuous network connectivity) and reassigning work units when a given node fails to report its results in expected time

The impacts of trust and availability on performance and development difficulty can influence the choice of whether to deploy onto a dedicated cluster, to idle machines internal to the developing organization, or to an open external network of volunteers or contractors. In many cases, the participating nodes must trust the central system not to abuse the access that is being granted, by interfering with the operation of other programs, mangling stored information, transmitting private data, or creating new security holes. Other systems employ measures to reduce the amount of trust "client" nodes must place in the central system such as placing applications in virtual machines.

Public systems or those crossing administrative domains (including different departments in the same organization) often result in the need to run on heterogeneous systems, using different operating systems and hardware architectures. With many languages, there is a trade off between investment in software development and the number of platforms that can be supported (and thus the size of the resulting network). Cross-platform languages can reduce the need to make this trade off, though potentially at the expense of high performance on any given node (due to run-time interpretation or lack of optimization for the particular platform). There are diverse scientific and commercial projects to harness a particular associated grid or for the purpose of setting up new grids. BOINC is a common one for various academic projects seeking public volunteers. more are listed at the end of the article.

In fact, the middleware can be seen as a layer between the hardware and the software. On top of the middleware, a number of technical areas have to be considered, and these may or may not be middleware independent. Example areas include SLA management, Trust and Security, Virtual organization management, License Management, Portals and Data Management. These technical areas may be taken care of in a commercial solution, though the cutting edge of each area is often found within specific research projects examining the field

## Market segmentation of the grid computing market

For the segmentation of the grid computing market, two perspectives need to be considered: the provider side and the user side:

### The provider side

The overall grid market comprises several specific markets. These are the grid middleware market, the market for grid-enabled applications, the utility computing market, and the software-as-a-service (SaaS) market.

Grid middleware is a specific software product, which enables the sharing of heterogeneous resources, and Virtual Organizations. It is installed and integrated into the existing infrastructure of the involved company or companies, and provides a special layer placed among the heterogeneous infrastructure and the specific user applications. Major grid middlewares are Globus Toolkit, gLite, and UNICORE.

Utility computing is referred to as the provision of grid computing and applications as service either as an open grid utility or as a hosting solution for one organization or a VO. Major players in the utility computing market are Sun Microsystems, IBM, and HP.

Grid-enabled applications are specific software applications that can utilize grid infrastructure. This is made possible by the use of grid middleware, as pointed out above.

Software as a service (SaaS) is "software that is owned, delivered and managed remotely by one or more providers." (Gartner 2007) Additionally, SaaS applications are based on a single set of common code and data definitions. They are consumed in a one-to-many model, and SaaS uses a Pay As You Go (PAYG) model or a subscription model that is based on usage. Providers of SaaS do not necessarily own the computing resources themselves, which are required to run their SaaS. Therefore, SaaS providers may draw upon the utility computing market. The utility computing market provides computing resources for SaaS providers.

### The user side

For companies on the demand or user side of the grid computing market, the different segments have significant implications for their IT deployment strategy. The IT deployment strategy as well as the type of IT investments made are relevant aspects for potential grid users and play an important role for grid adoption.

## CPU scavenging

**CPU-scavenging**, **cycle-scavenging**, **cycle stealing**, or **shared computing** creates a "grid" from the unused resources in a network of participants (whether worldwide or internal to an organization). Typically this technique uses desktop computer instruction cycles that would otherwise be wasted at night, during lunch, or even in the scattered seconds throughout the

day when the computer is waiting for user input or slow devices. In practice, participating computers also donate some supporting amount of disk storage space, RAM, and network bandwidth, in addition to raw CPU power.

Many Volunteer computing projects, such as BOINC, use the CPU scavenging model. Since nodes are likely to go "offline" from time to time, as their owners use their resources for their primary purpose, this model must be designed to handle such contingencies.

## History

The term *grid computing* originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid. The power grid metaphor for accessible computing quickly became canonical when Ian Foster and Carl Kesselman published their seminal work, "The Grid: Blueprint for a new computing infrastructure" (2004).

CPU scavenging and volunteer computing were popularized beginning in 1997 by distributed.net and later in 1999 by SETI@home to harness the power of networked PCs worldwide, in order to solve CPU-intensive research problems.

The ideas of the grid (including those from distributed computing, object-oriented programming, and Web services) were brought together by Ian Foster, Carl Kesselman, and Steve Tuecke, widely regarded as the "fathers of the grid". They led the effort to create the Globus Toolkit incorporating not just computation management but also storage management, security provisioning, data movement, monitoring, and a toolkit for developing additional services based on the same infrastructure, including agreement negotiation, notification mechanisms, trigger services, and information aggregation. While the Globus Toolkit remains the de facto standard for building grid solutions, a number of other tools have been built that answer some subset of services needed to create an enterprise or global grid.

In 2007 the term cloud computing came into popularity, which is conceptually similar to the canonical Foster definition of grid computing (in terms of computing resources being consumed as electricity is from the power grid). Indeed, grid computing is often (but not always) associated with the delivery of cloud computing systems as exemplified by the AppLogic system from 3tera

## Fastest virtual supercomputers[edit]

- As of April 2013, Folding@home – 11.4 x86-equivalent (5.8 "native") PFLOPS.
- As of March 2013, BOINC – processing on average 9.2 PFLOPS.
- As of April 2010, MilkyWay@Home computes at over 1.6 PFLOPS, with a large amount of this work coming from GPUs.[9]
- As of April 2010, SETI@Home computes data averages more than 730 TFLOPS

- As of April 2010, Einstein@Home is crunching more than 210 TFLOPS
- As of June 2011, GIMPS is sustaining 61 TFLOPS

## Projects and applications

*Main article: List of distributed computing projects*

Grids computing offer a way to solve Grand Challenge problems such as protein folding, financial modeling, earthquake simulation, and climate/weather modeling. Grids offer a way of using the information technology resources optimally inside an organization. They also provide a means for offering information technology as a utility for commercial and noncommercial clients, with those clients paying only for what they use, as with electricity or water.

Grid computing is being applied by the National Science Foundation's National Technology Grid, NASA's Information Power Grid, Pratt & Whitney, Bristol-Myers Squibb Co., and American Express.

One cycle-scavenging networks is SETI@home, which was using more than 3 million computers to achieve 23.37 sustained teraflops (979 lifetime teraflops) as of September 2001

As of August 2009 Folding@home achieves more than 4 petaflops on over 350,000 machines.

The European Union funded projects through the framework programmes of the European Commission. BEinGRID (Business Experiments in Grid) was a research project funded by the European Commission as an Integrated Project under the Sixth Framework Programme (FP6) sponsorship program. Started on June 1, 2006, the project ran 42 months, until November 2009. The project was coordinated by Atos Origin. According to the project fact sheet, their mission is "to establish effective routes to foster the adoption of grid computing across the EU and to stimulate research into innovative business models using Grid technologies". To extract best practice and common themes from the experimental implementations, two groups of consultants are analyzing a series of pilots, one technical, one business. The project is significant not only for its long duration, but also for its budget, which at 24.8 million Euros, is the largest of any FP6 integrated project. Of this, 15.7 million is provided by the European commission and the remainder by its 98 contributing partner companies. Since the end of the project, the results of BEinGRID have been taken up and carried forward by IT-Tude.com.

The Enabling Grids for E-sciencE project, based in the European Union and included sites in Asia and the United States, was a follow-up project to the European DataGrid (EDG) and evoled into the European Grid Infrastructure. This, along with the LHC Computing Grid (LCG), was developed to support experiments using the CERN Large Hadron Collider. The A list of active sites participating within LCG can be found online as can real time

monitoring of the EGEE infrastructureThe relevant software and documentation is also publicly accessible. There is speculation that dedicated fiber optic links, such as those installed by CERN to address the LCG's data-intensive needs, may one day be available to home users thereby providing internet services at speeds up to 10,000 times faster than a traditional broadband connection.

The distributed.net project was started in 1997. The NASA Advanced Supercomputing facility (NAS) ran genetic algorithms using the Condor cycle scavenger running on about 350Sun Microsystems and SGI workstations.

In 2001, United Devices operated the United Devices Cancer Research Project based on its Grid MP product, which cycle-scavenges on volunteer PCs connected to the Internet. The project ran on about 3.1 million machines before its close in 2007.

As of 2011, over 6.2 million machines running the open-source Berkeley Open Infrastructure for Network Computing (BOINC) platform are members of the World Community Grid, which tops the processing power of the current fastest supercomputer system (China's Tianhe-I).

## Definition

Today there are many definitions of *grid computing*:

- In his article "What is the Grid? A Three Point Checklist", Ian Foster lists these primary attributes:
  - Computing resources are not administered centrally.
  - Open standards are used.
  - Nontrivial quality of service is achieved.

- Plaszczak/Wellner define grid technology as "the technology that enables resource virtualization, on-demand provisioning, and service (resource) sharing between organizations."
- IBM defines grid computing as "the ability, using a set of open standards and protocols, to gain access to applications and data, processing power, storage capacity and a vast array of other computing resources over the Internet. A grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of resources distributed across 'multiple' administrative domains based on their (resources) availability, capacity, performance, cost and users' quality-of-service requirements".
- An earlier example of the notion of computing as utility was in 1965 by MIT's Fernando Corbató. Corbató and the other designers of the Multics operating system envisioned a computer facility operating "like a power company or water company".

- Buyya/Venugopal define grid as "a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed autonomousresources dynamically at runtime depending on their availability, capability, performance, cost, and users' quality-of-service requirements".
- CERN, one of the largest users of grid technology, talk of **The Grid**: "a service for sharing computer power and data storage capacity over the Internet.

Grids can be categorized with a three stage model of departmental grids, enterprise grids and global grids. These correspond to a firm initially utilising resources within a single group i.e. an engineering department connecting desktop machines, clusters and equipment. This progresses to enterprise grids where nontechnical staff's computing resources can be used for cycle-stealing and storage. A global grid is a connection of enterprise and departmental grids that can be used in a commercial or collaborative manner.