Títol: Record Linkage Paral·lel

Volum: 1

Alumne: Joan Guisado Gámez

Director/Ponent: Josep Lluís Larriba Pey

Departament: Arquitectura de Computadors

Data: 26 de gener del 2009

Facultat d'Informàtica de Barcelona
Universitat Politàcnica de Catalunya

PROJECTE FINAL DE CARRERA

Record Linkage Paral·lel

Alumne:
Joan Guisado Gámez

Director:
Josep Ll. Larriba Pey

Agraïments

En primer lloc vull agrair a totes aquelles persones que, en major o menor mesura, han fet possible la realització d'aquest projecte.

Al Josep Lluís Larriba Pey, el meu Director, per donar-me la possibilitat de dur-lo a terme, per guiar-me i ajudar-me quan ha calgut.

A l'Arnau Prat Pérez, amic i company de DAMA-UPC, per la seva impagable ajuda i suport en tot el projecte.

Al Francesc Escalé Claveras, desenvolupador de DAMA-UPC, que m'ha ajudat a desxifrar el codi de DAURUM.

A tots els meus companys i ex-companys de DAMA-UPC que han omplert de bons moment la realització del PFC.

Als meus amics, amb els qui comparteixo d'altres projectes en els que cada dia hi crec més.

A la Isabel i l'Ariel, els meus pares, pel seu recolzament moral i econòmic, sense els que hagués estat impossible dur a terme aquesta carrera. A la meva família, en especial a la meva tia avia, la Isabel, i els meus avis, la Irene, la Dolors i el Josep, que sense entendre què és això al que em vull dedicar han cregut en mi i m'han donat suport.

Índex

Capítol 1

Idea del projecte

En les darreres tres dècades la societat ha viscut una implantació, sovint descontrolada, de les eines informàtiques en la seva vida diaria. D'un moment a un altre totes les organitzacions mitjanes i grans han passat a tenir les seves bases de dades amb la informació dels seus usuaris. És possible imaginar, per exemple, un centre mèdic d'un poble, als anys '80. El director d'aleshores va decidir que seria molt útil disposar d'una base de dades dels pacients on hi constessin diferents informacions. Amb el pas del temps la base de dades ha anat creixent degut a l'ús intensiu que se n'ha fet. Arribat el moment de fer un anàlisi de les dades s'ha trobat que la base de dades no era consistent. S'han trobat diferents entrades que corresponen a la mateixa persona, però amb petites diferències a l'hora d'escriure el nom.

Aquest fet pot ser degut a que l'encarregat de dissenyar la base de dades no tingués en compte diferents aspectes com, per exemple, l'ús d'un identificador únic per pacient.

Ens trobem, doncs, davant d'un repte que consisteix en trobar aquelles entrades d'una base de dades que corresponen a una mateixa entitat. Es pot veure que realitzar aquest procés de manera manual esdevé, de seguida, un problema intractable. És aquí que apareixen les tècniques que s'anomenen **Record Linkage** (RL). Aquestes tècniques tenen com a objectiu emparellar aquelles entrades que tenen un cert grau de similitud. En finalitzar el procés es té un conjunt d'entrades emparellades que cal que un expert decideixi si corresponen, realment, a una mateixa entitat.

El Record Linkage resulta una eina molt útil per netejar una base de dades, i aquelles entrades que corresponen a una mateixa entitat, però aquesta no és la única utilitat que té. Avui en dia existeix una realitat i és que es tendeix a globalitzar la informació. Les organitzacions comparteixen informació o fins i tot es fusionen. La conseqüència final, en qualsevol cas, acaba sent la mateixa: cal fusionar o mantenir sota una mateixa perspectiva les diferents bases de dades que corresponen a un mateix domini. És provable que aquestes bases de dades continguin informació sobre la mateixa entitat i cal, per tant, que a l'hora de fusionar-les es tingui en compte i es detectin aquests casos per tal de que la base de dades resultant de la fusió no sigui inconsistent.

Finalment, per posar un altre exemple de la utilitat del RL, cal tenir en compte que en l'actualitat existeix una necessitat real per fer anònimes les bases de dades. És a dir, ser capaços de mantenir les propietats estadístiques de les bases de dades sense que hom pugui ser capaç de relacionar cadascuna de les entrades de la base de dades amb la entitat real a la que corresponen. Per això fa anys que s'està fent recerca en tècniques d'anonimització. Una manera per avaluar aquestes tècniques és emprant RL. Si el procés és capaç de relacionar les entitats anonimitzades amb les reals significa que el mètode d'anonimització és poc eficaç.

L'Annex B justament és un article que es va presentar en el congrés que duu per nom

Privacy in Statistical Databases - 2008 on es justifica la paral·lelització del procés justament sota aquesta utilització de RL.

El funcionament del procés de Record Linkage és el següent. A partir d'un fitxer de registres com el que es pot veure en la Figura 1.1 es comparen els valors dels seus atributs, com ara Nom, Cognom, Adreça, etc. per trobar les similituds entre ells.

Nom	Cognom 1	Cognom 2	Edat	Sexe	Població	País
Jesús	Santana	Masdeu	33	M	Arbúcies	Catalunya
Josep M.	Subirats	Puigfalcó	65	M	Barcelona	Catalunya
Josep Maria	Sovirats	Puigfalcó	65	M	Barcelona	Catalunya
Ramon	Solé	Recasens	83	D	Vic	Catalunya
Ramona	Solé	Recasens	93	D	Vic	Catalunya
Jesep M.	Subirats	Puigfalcó	65	M	Barcelona	Catalunya
Roger	Subirats	Garcia	21	M	Girona	Catalunya
Rosalia	Subirats	Gracia	23	D	Igualada	Catalunya

Figura 1.1: Fitxer de registres amb set atributs: Nom, Cognom 1, Cognom 2, Edat, Sexe, Població i País, i vuit registres.

Es pot veure, doncs, que la comparació entre els registres és un procés de complexitat quadràtica, $O(N^2)$, on N és el nombre de registres de la base de dades amb la que s'està treballant. Això és així perquè cada registre s'ha de comparar amb la resta de registres. Cal tenir en compte que existeixen grans BD i que sovint el temps esdevé una restricció important ja que aturar el sistema de gestió de la informació pot voler dir deixar d'oferir els serveis que la organització en qüestió dóna als seus usuaris. És a dir, el procés de Record Linkage és un procés lent que té lligades unes restriccions de temps importants i, per tant, cal realitzar-lo el més ràpid possible. Cal doncs fer servir mètodes que redueixin el temps de càlcul. Una primera mesura és que el fitxer de registres amb el que es treballa ha d'estar carregat completament en la memòria RAM, d'això se'n diu treballar **in-core**. Existeixen maneres de treballar **out-of-core**, és a dir treballar amb els registres fora de la memòria, llegint-los directament del fitxer, que es troba en el disc dur, a mesura que es necessiten. Això és, però, extremadament lent i donat que el sistema de RL ja és lent de per si, aquesta possibilitat no es sol contemplar. Notis, però, que això afegeix una restricció: el conjunt de registres amb el que es treballa ha de cabre en la memòria del sistema en que s'està executant.

És necessari, doncs, trobar maneres d'accelerar el procés de Record Linkage. En la literatura es poden trobar mètodes que permeten reduir la complexitat del problema. Aquests mètodes, però, tot i accelerar el procés, sovint no assoleixen les restriccions de temps que la realitat imposa.

1.1 Motivació

Tal i com s'ha dit, el Record Linkage és capaç de trobar aquelles entrades, o registres, de la base de dades que poden correspondre a una mateixa entitat real. El Record Linkage, però, és lent, i cal, per tant, trobar maneres d'accelerar-lo tenint en compte que les dades han d'estar en memòria.

És clar que si s'empra un gran ordinador, un supercomputador, que efectua milers de millions de càlculs per segon i amb quantitats molt grans de memòria, les tècniques existents

poden ser més que sobrades per realitzar el procés de RL. Emprar aquesta tecnologia, però, no està a l'abast de tothom; poder sí en les mans d'algunes grans organitzacions, però les mitjanes o d'altres sense ànim de lucre, com els centres d'estudis estadístiques, difícilment podran accedir a aquests supercomputadors. Què passa, doncs, si no es disposa d'aquesta tecnologia?

Aquest projecte se centra en la situació de institucions que, com ara centres estadístics, tenen la necessitat d'executar processos de Record Linkage, però no d'altres processos costosos de càlcul i per tant, no té sentit comprar grans computadors que realitzin tasques poc freqüents, que per altra banda, és probable que no puguin tenir a l'abast.

Avui en dia és quelcom normal, en les organitzacions en les que se centra el projecte, tenir una xarxa amb els ordinadors de sobretaula. Aquests xarxa és usada pels seus empleats per dur a terme les seves tasques laborals, però quan la jornada de feina acaba els ordinadors resten sense fer res. El que es proposa és, doncs, usar aquesta xarxa d'ordinadors, quan estigui desocupada, per accelerar el procés de Record Linkage. La idea és que aquests ordinadors treballin conjuntament per executar el procés de manera que el temps d'execució es vegi reduït. Notis que aquesta proposta també permet treballar amb més registres ja que cada ordinador té la seva pròpia memòria i per tant, la memòria total és més gran que si només es tingués un únic ordinador.

1.2 Conceptes bàsics: Paral·lelisme

La paral·lelització és una tècnica del camp de la computació que té com a objectiu, **donada una determinada tasca dividir-la en d'altres més petites** que es poden realitzar de manera concurrent, és a dir, a la vegada. La finalitat és reduir el temps de càlcul distribuint la càrrega de feina entre els diferents processadors disponibles. Es poden distingir dos grans blocs d'ordinadors paral·lels. Per una banda, els que tenen la memòria compartida, és a dir, un mateix espai d'adreces compartit al que hi poden accedir tots els processadors mitjançant un bus com el que es pot veure en la Figura 1.2.

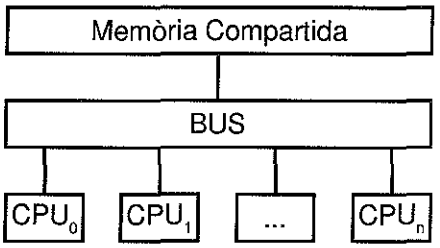

Figura 1.2: Esquema d'un model d'ordinador amb memòria compartida.

Aquest model comporta una sèrie de dificultats, com poden ser l'accés simultani a la memòria, o la poca escalabilitat, ja que el fet d'augmentar el nombre de processadors pot crear un coll d'ampolla a l'hora d'accedir a la memòria. El punt a favor d'aquest tipus d'ordinadors és que els programes que executen són força fàcils de programar. Sovint per dur a terme aquest tipus de paral·lelitzacions s'empren les interfícies **POSIX Threads-PThreads** [7] o **OpenMP** [8], que són les més populars.

Per una altra banda hi ha els sistemes paral·lels de memòria distribuïda. Es tracta d'un conjunt d'ordinadors independents, anomenats nodes, que estan connectats entre si

mitjançant una xarxa com la que es pot veure en la Figura 1.3. Cada processador està en un node diferent del sistema i per tant té la seva pròpia memòria local. Els processadors poden compartir informació mitjançant missatges, és a dir, si un processador A vol les dades contingudes en la memòria d'un altre processador B haurà d'enviar un missatge demanant-les-hi. Per l'altra banda, quan B vulgui respondre amb la informació sol·licitada també ho haurà de fer mitjançant un missatge. Aquest tipus de comunicació es coneix com a **pas de missatges**.

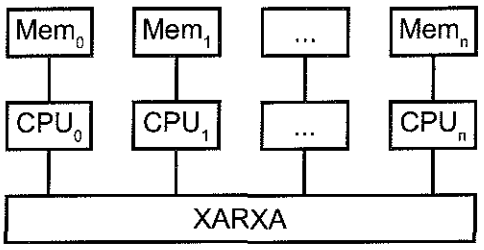

Figura 1.3: Esquema d'un model d'ordinador amb memòria distribuïda.

El principal avantatge d'aquest model és l'escalabilitat que ofereix ja que a mesura que creix la demanda de recursos es pot afegir més memòria i processadors. Els inconvenients són la dificultat a l'hora de programar i la lentitud en el pas d'informació, comparat amb els sistemes de memòria compartida. La interfície per excel·lència per dur a terme aquesta mena de paral·lelitzacions és **MPI - Message Passing Interface** [13], encara que n'hi ha d'altres com **PVM - Parallel Virtual Machine** [5], més en desús actualment. Per altra banda cal tenir en compte que cadascun d'aquests nodes pot ser un ordinador de memòria compartida, permetent així una paral·lelització a dos nivells.

1.3 Objectius

En aquesta secció es fan explícits, i es detallen, els objectius que s'han anat dibuixant en les seccions anteriors. Els objectius estan classificats de més generals (més exteriors), a més específics (més interiors).

1 Accelerar el procés de Record Linkage mitjançant tècniques de computació paral·lela.

- 1.1 Aprendre a utilitzar MPI.
- 1.2 Implementar una paral·lelització distribuïda.
- 1.3 Utilitzar ordinadors comuns, de sobretaula.
- 1.4 Implementar una paral·lelització de memòria compartida a nivell de node.
- 1.5 Dur a terme una implementació conscient de les restriccions de memòria.
- 1.6 Utilitzar els mètodes que s'usen per accelerar el RL seqüencial.
  - 1.6.1 Entendre el funcionament dels mètodes.
  - 1.6.2 Analitzar el que suposa paral·lelitzar aquests mètodes.

2 Publicar un article.

- 2.1 Aprendre a escriure un article per a una conferència.

2.2 Explicar en un article aquells aspectes que poden tenir un interès en el camp de la recerca.

2.3 Aprendre a presentar un article de recerca davant d'una audiència.

3 Documentar el Projecte Final de carrera.

3.1 Escriure una memòria clara i entenedora que permeti entendre fàcilment la feina feta en el decurs del projecte.

3.2 Fer una presentació del projecte clara, conscisa i entenedora.

1.4 Estructura del document

El document s'estructurarà com segueix. En el Capítol 2 es descriurà amb més detall que és el Record Linkage i els problemes de rendiments que comporta la seva aplicació. S'estudiaran també quines són les solucions més comunes que es poden trobar en la literatura per accelerar el procés de Record Linkage. En el Capítol 3 s'estudiarà una altra solució que permet també accelerar el procés RL i que és la que es farà servir com a base de la paral·lelització. Es veuran, en el Capítol 4, les diferents propostes de paral·lelitzacions que s'han fet. L'estudi i anàlisi d'aquestes paral·lelitzacions que s'han proposat es farà en el Capítol 5. Es veurà també, en el Capítol 6, informació sobre la gestió del projecte, planificació i costos. Finalment, en el Capítol 7 s'analitzaran quines són les conclusions que se'n poden extreure i quines són aquelles tasques, o vies a explotar, de cara a realitzar una continuació del projecte.

Capítol 2

Record Linkage

En aquest segon capítol s'explica què és el Record Linkage, quins són els reptes que presenta a nivell de rendiment de les aplicacions i algunes de les propostes que es poden trobar en la literatura per augmentar-ne el rendiment.

2.1 Contextualització

El procés de Record Linkage s'engloba dins del conjunt de tècniques de re-identificació. Aquestes són una classe específica de tècniques que s'apliquen al camp de la gestió de la informació i estan centrades en establir relacions entre les diferents entitats que hi ha en les diferents fonts de dades. Per tant, obtenir relacions entre entitats té sentit, entre d'altres, en aquests escenaris:

- **Schema matching** [20]. Consisteix en, donats dos esquemes diferents, trobar les relacions que hi pot haver entre els atributs dels diferents esquemes. Aquest és un problema bàsic en moltes aplicacions basades en dades.

- **Data integration** [10]. Té com a fi crear una vista integrada de diferents fonts de dades aparentment incompatibles. La incompatibilitat ve dondada per les diferents percepcions i requeriments que sovint fan expressar una mateixa informació de múltiples maneres.

- **Data cleansing** [11]. L'objectiu és detectar i eliminar els errors i inconsistències de les dades de manera que se'n millori la qualitat. Els errors i inconsistències més comuns poden ser de manca de dades, dades invalides o paraules incompletes.

- **Object integration** [19]. Consisteix en establir relacions entre objectes que tenen propietats, o comportaments, similars.

- **Master data management** [18]. Té com a objectiu nodrir de processos que permetin recollir, agregar, creuar, consolidar i assegurar la qualitat, de la informació de les organitzacions per garantir-ne el control i permetre'n un ús controlat.

El Record Linkage permet fer re-identificació i freqüentment s'utilitza en la neteja de les dades [21] i en la integració de conjunts de dades distribuïts i heterogenis. L'objectiu, doncs, acaba sent identificar, d'entre un conjunt de registres, quins corresponen a una mateixa entitat física.

2.2 Fases del Record Linkage

Es pot entendre el RL com un conjunt de diferents fases tal i com es mostra en la Figura 2.1.

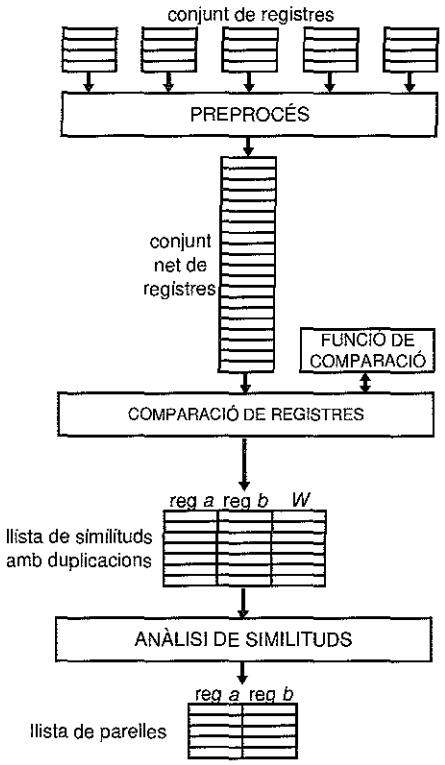

Figura 2.1: El procés de Record Linkage dividit en les fases que el conformen.

El procés comença amb una primera fase de **preprocés**. En aquesta fase es concatenen els diferents fitxers de registres amb els que es vol treballar, es normalitzen, es netegen individualment els valors i es carreguen els registren en memòria. Un cop s'ha realitzat aquesta fase es té un únic fitxer de registres amb els valors nets i normalitzats.

La segona fase consisteix en **comparar els registres**. Cada registre es compara amb la resta de registres mitjançant una **funció de comparació** de registres. La funció de comparació retorna un grau de similitud W. Donats dos registres x i y compostos per n atributs, W es calcula així: $W(x,y) = \sum_{i=1}^{n} W_i(x_i, y_i)$. És a dir, es calculen els graus de similitud parcials, a partir de la comparació entre atributs, i s'usen després per calcular el grau de similitud global entre els dos registres.

Només aquelles parelles de registres tals que el seu grau de similitud W sigui superior a un llindar mínim de similitud T es classificaran com a registres que potencialment corresponen a una mateixa entitat.

És interessant definir, ara, dos conceptes. El **recall**, que correspon a les similituds trobades, respecte a les que existeixen realment i la **precisió**, que correspon a la fracció de similituds reals respecte les similituds trobades.

Així doncs, com més petit sigui el llindar de similitud, T, s'obtindrà un major recall però una menor precisió.

La tercera fase consisteix en fer una **anàlisi manual de les similituds** que ha trobat el procés. Per tant és important triar el llindar mínim de similitud perquè el procés trobi el

màxim nombre de similituds, però alhora redueixi les falses similituds, és a dir, obtingui el màxim recall i precisió.

D'entre totes aquestes fases la més costosa, computacionalment parlant, és la fase en la que es comparen els registres. Si es té en compte que cada registre s'ha de comparar amb la resta de registres, es pot afirmar que el cost d'aquesta fase és quadratic, $O(N^2)$, on N és el nombre de registres. Per altra banda, la resta de les fases tenen una complexitat lineal, $O(N)$. Així doncs, per augmentar el rendiment del procés cal atacar directament a la fase en la que es comparen els registres.

A partir d'aquest moment, si no es diu el contrari, quan es faci referència a Record Linkage, s'estarà fent referència a l'etapa on es produeix la comparació de registres. Això és així perquè aquesta és l'etapa més costosa i en la que se centra el projecte. D'aquesta manera es guanyarà en comprensibilitat i es facilitarà la lectura del document.

La recerca per accelerar el procés de Record Linkage fa anys que existeix. Tots els esforços s'han concentrat a reduir la complexitat del procés i en aquest sentit es pot trobar en la literatura el que s'anomena mètodes de blocking.

2.3 Mètodes de blocking

Com ja s'ha explicat l'aplicació del Record Linkage és molt costosa. Això ve donat perquè cada registre s'ha de comparar amb la resta de registres. Els mètodes de blocking s'empren, doncs, per reduir el nombre de registres amb els que un determinat registre s'ha de comparar. Els mètodes de blocking treballen amb blocs de registres. Les comparacions entre els registres es fan a nivell de bloc a diferència que fins ara, en que les comparacions es feien a nivell de tot el conjunt de registres. Els dos mètodes clàssics són: **Standard Blocking** [2] [10] i **Sliding Window** [14].

2.3.1 Standard Blocking

El mètode Standard Blocking agrupa els registres que comparteixen una mateixa **clau de bloc**. Les claus es defineixen a partir dels atributs dels registres. Un exemple de clau és: *aquells registres tals que tinguin iguals els quatre primers caràcters de l'atribut Nom.* Una clau no té perquè ser composta per un únic atribut, es podrien crear els blocs, per exemple, amb: *aquells registres tals que tinguin iguals l'atribut Adreça i l'atribut Edat.*

Per donar una idea del funcionament del mètodes es pot pensar en aplicar Record Linkage a la taula de la Figura 1.1. Sense emprar cap mètode s'haurien de fer vint-i-vuit comparacions entre registres. Si s'aplica Standard Blocking, emprant com a clau *aquells registres tals que tinguin iguals els quatre primers caràcters de l'atribut Nom* s'obtindrien sis blocs, tal i com es veu en la Figura 2.2. Tenint en compte que cada registre es compara amb la resta de registres del mateix bloc es pot dir que el mètode de Standard Blocking amb els blocs de la Figura 2.2 i la clau de bloc esmentada, faria dues comparacions entre registres.

Cal tenir en compte, a l'hora de triar les claus, que cal buscar un equilibri entre el recall i el cost de l'algoritme. Per exemple, si es tria com a clau *aquells registres tals que tinguin iguals l'atribut Sexe* hi haurà dos grans blocs, per tant caldrà efectuar moltes comparacions i no s'aconseguirà una gran millora en el temps d'execució respecte a la versió que no fa servir Standard Blocking. Per una altra banda, si s'escull una clau que creï molts blocs, i molt petits, segurament es deixaran d'efectuar comparacions entre registres que corresponen a la mateixa entitat.

Jesús	Santana	Masdeu	33	M	Arbúcies	Catalunya
Jesep M.	Subirats	Puigfalcó	65	M	Barcelona	Catalunya
Josep Maria	Sovirats	Puigfalcó	65	M	Barcelona	Catalunya
Josep M.	Subirats	Puigfalcó	65	M	Barcelona	Catalunya
Roger	Subirats	Garcia	21	M	Girona	Catalunya
Rosalia	Subirats	Gracia	23	D	Igualada	Catalunya
Ramona	Solé	Recasens	93	D	Vic	Catalunya
Ramon	Solé	Recasens	83	D	Vic	Catalunya

Figura 2.2: Blocs de registres resultants d'haver aplicat Standard Blocking a la Taula de la Figura 1.1 emprant la clau: *quatre primers caràcters de l'atribut Nom.*

Una altra consideració que cal tenir en compte a l'hora d'escollir la clau és les característiques que tenen els errors dels atributs escollits. Per obtenir bons recalls es recomana emprar l'atribut que es consideri menys propens a tenir errors [4]. En la Figura 2.2, en la que s'ha fet servir una clau basada en l'atribut Nom, s'observa que la clau de bloc ha fet que es creessin sis blocs. S'observa, però, que l'únic registre que conforma el segon dels blocs, correspon a la mateixa entitat física que els registres del tercer bloc. Així doncs s'estan deixant de fer unes comparacions que tindrien com a conseqüència trobar més similituds. Per tal de que aquestes comparacions s'acabin efectuant i, per tant, s'obtingui un recall més alt, es poden realitzar diferents iteracions del procés emprant, a cada iteració, diferents claus de blocs, de manera que els registres conformin blocs diferents a cada iteració. Com a conseqüència de realitzar diferents iteracions es pot detectar múltiples vegades la mateixa similitud. Això fa necessari una fase d' **eliminació de les similituds repetides.**

S'ha dit que el mètode de Standard Blocking redueix la complexitat del procés de RL. Cal veure, doncs, quina és la complexitat del procés si s'aplica. Assumint que hi ha N registres, i que el mètode de blocking retorna b blocs de la mateixa mida, es pot dir doncs que la mida dels blocs és $B = N/b$. Per tant, la complexitat de l'algoritme passa a ser $O(B \cdot N)$. Com que difícilment tots els blocs seran de la mateixa mida, es pot considerar que la complexitat és $O(B' \cdot N)$, on B' és la mida del bloc més gran.

2.3.2 Sliding Window

El mètode Sliding Window, o també Sorted Neighborhood, ordena els registres segons una **clau d'ordenació**. Un cop feta la ordenació defineix una **window**, que no és més que un conjunt de w registres i que a partir d'ara s'anomenarà **finestra**. La finestra es va desplaçant registre a registre, tal i com es pot veure en la Figura 2.3. De la mateixa manera que en Standard Blocking, per definir les claus d'ordenació s'empren un o més atributs. El primer registre de la finestra es compara amb la resta de registres que cauen dins de la mateixa finestra.

L'efectivitat del mètode dependrà de la qualitat de les claus d'ordenació triades. Si no s'escull una bona clau pot resultar una ordenació que distribueixi els registres que corresponen

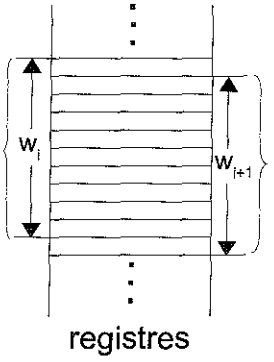

Figura 2.3: Comportament de la finestra, amb w=10, en un procés de RL. Es mostra la finestra i-èssima, i finestra $i+1$ que és la següent.

a una mateixa entitat en diferents finestres, i per tant no se'ls identificaria com a registres similars. Per triar una bona clau cal tenir en compte aquells atributs amb una capacitat de discriminació més alta, i menys propensos a tenir errors [4]. També és molt important escollir el nombre de registres, w, que conformen les finestres, doncs si w és molt gran el mètode serà ineficient, però per altra banda, si w és molt petit es deixaran de comparar registres que podrien acabar corresponent a la mateixa entitat. Per obtenir un recall més alt es poden realitzar diferents iteracions, emprant a cada iteració diferents criteris d'ordenació. Com a conseqüència de realitzar diferents iteracions, es pot detectar múltiples vegades la mateixa similitud. Això fa necessari una fase d'**eliminació de les similituds repetides**, de la mateixa manera que passava amb Standard Blocking.

Si s'analitza la complexitat del mètode sobre un conjunt de N registres es veu que el procés d'ordenació dels registres segons la clau d'ordenació té una complexitat $O(N \cdot \log(N))$, la de detecció de les similituds és $O(w \cdot N)$, on w és la mida de la finestra i finalment la de l'eliminació de similituds replicades que és $O(N)$. Per tant la complexitat de l'algoritme és $O(w \cdot N)$ si $w > \log(N)$, altrament la complexitat és $O(N \cdot \log(N))$.

2.3.3 Conclusions dels mètodes de blocking

S'ha vist que els mètodes de blocking serveixen per reduir el nombre de comparacions i, per tant, per reduir la complexitat del procés de RL. Els mètodes de blocking, però, fan que el recall sigui més baix i per tal de fer-lo créixer es fan diverses iteracions del procés. Això té com a conseqüència trobar la mateixa similitud diverses vegades, i per tant cal una nova fase d'eliminació de les similituds repetides. Així doncs, la fase de comparació de registres de la Figura 2.1 ha passat a estar formada per tres fases que es poden veure en la Figura 2.4.

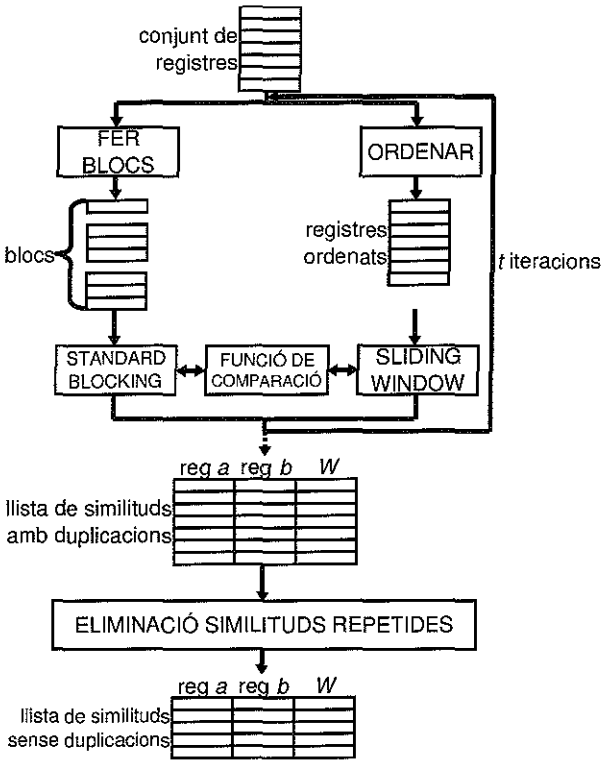

Figura 2.4: Es mostren les tres etapes del Record Linkage suposant un mètode de blocking. La primera etapa de fer els blocs o ordenar els registres, la segona etapa fer les comparacions dels registres, ja sigui amb un mètode de Standard Blocking o de Sliding Window respectivament i la tercera etapa, després de fer t iteracions de les dues anteriors, eliminar similituds repetides.

És interessant, abans de continuar, fer-se una idea del funcionament, a la pràctica, dels mètodes de blocking, sobretot pel que fa al recall que s'obté. A continuació es mostra un petit experiment. Quatre configuracions diferents, tres amb Standard Blocking i una amb Sliding Window, cadascuna fa quatre iteracions. A la primera iteració es farà servir com a clau el Nom, en la segona el Primer Cognom, en la tercera el Segon Cognom i en la quarta la Data de Neixament. El nombre de caràcters que s'ha emprat per fer els blocs, o la ordenació, en cadascuna de les iteracions es mostra en la Taula 2.1.

	Iteració			
Mètode i Configuració	Nom	Primer Cognom	Segon Cognom	Data de naixament
	Nombre de caràcters			
S. Blocking-A	3	2	2	M/A
S. Blocking-B	4	3	3	D/M/A
S. Blocking-C	5	4	4	D/M/A
S. Window-A	tots	tots	tots	D/M/A

Taula 2.1: configuracions de Standard Blocking i Sliding Window.

En la Taula 2.2 es pot veure el recall que ha obtingut cada configuració amb diferents

quantitats de registres. En el cas d'aplicar Sliding Window també es mostra la mida de la finestra que s'ha emprat. S'observa que els recalls obtingut s'apropen molt al 100%.

Mètode i Configuració	Número de registres		
	10^4	10^5	10^6
	Recall		
S. Blocking-A	99.21%	98.90%	98.85%
S. Blocking-B	99.19%	98.87%	98.81%
S. Blocking-C	99.11%	98.80%	98.71%
S. Window-A	98.96%	98.21%	96.71%
	Mida de la finestra		
	30	90	270

Taula 2.2: Recalls per diferents configuracions de Standard Blocking i Sliding Window.

Malgrat l'increment de rendiment que assoleixen els mètodes de blocking, el RL es segueix considerant un mètode molt costós. Això és degut a que per cada comparació entre registres cal fer n comparacions entre els valors dels atributs que els conformen. En el següent capítol s'explica un mètode que té com a objectiu reduir el nombre d'aquestes comparacions.

Capítol 3

Tècniques de memoització

Els mètodes de blocking resulten útils per reduir la complexitat del problema de Record Linkage mitjançant la reducció del nombre de comparacions entre registres. Tot i així, aquests mètodes resulten insuficients per assolir els requeriments pel que fa les restriccions en el temps que existeixen avui en dia. En computació les tècniques de memoització s'empren per obtenir un major rendiment en l'execució dels programes. La tècnica consisteix en recordar els resultats dels càlculs que es van efectuant en el decurs del procés per així poder-se estalviar repetir els mateixos càlculs diverses vegades. A continuació es mostrarà com aplicar memoització a un procés de Record Linkage que utilitza, alhora, algun dels mètodes de blocking vistos fins el moment.

3.1 Introducció

Aquesta secció té com a objectiu demostrar que el procés de Record Linkage és un procés altament optimitzable a partir de l'aplicació de la tècnica de memoització. Primer es veurà d'una manera, informal i intuïtiva, per passar després a una justificació més formal.

Com ja s'ha explicat, per comparar dos registres es compararà atribut amb atribut. Hi ha atributs, com ara Nom, Cognom, Adreça, etc. els valors dels quals són **cadenes de caràcters**(també anomenats **strings**). A diferència de la comparació entre valors numèrics que són càlculs relativament senzills, la comparació entre els valors que són cadenes de caràcters són uns càlculs molt complexos. Tot i que hi ha diferents maneres per fer-ho aquest projecte suposarà que s'està emprant la distància de Levenshtein [17], que és un dels mètodes més utilitzats. En l'Annex C es pot veure com es calcula aquesta distància i comprovar que és un càlcul complex. Així doncs, la memoització en el procés de Record Linkage persegueix l'objectiu d'**estalviar-se tantes comparacions entre cadenes de caràcters com sigui possible.** A partir d'aquest moment, i per facilitar la lectura del document, **es suposarà que els atributs als quals es fa referència contenen cadenes de caràcters.**

En la Figura 3.1 es pot observar una finestra de mida 8 d'un conjunt de registres que han estat ordenats per l'atribut Població. S'observa que per calcular el grau de similitud entre els diferents registres, es comparen els atributs Nom, Cognom 1, Cognom 2, Població i País. Com ja s'ha explicat en la Secció 2.3, el primer registre de la finestra s'ha de comparar amb tota la resta de registres, en total amb set registres. Es pot veure que cal fer cinc comparacions per registre, una per cada atribut, en total trenta-cinc comparacions entre els valors dels atributs. Si es guarda el resultat de les comparacions el nombre de càlculs a efectuar es veu reduït de manera que només cal realitzar: sis comparacions entre els valors de l'atribut Nom, tres amb els de Cognom 1, quatre amb els de Cognom 2, quatre amb els de Població i una amb els de

País; en total divuit. És a dir, aplicant la tècnica de memoització s'ha reduït a gairebé la meitat el nombre de càlculs.

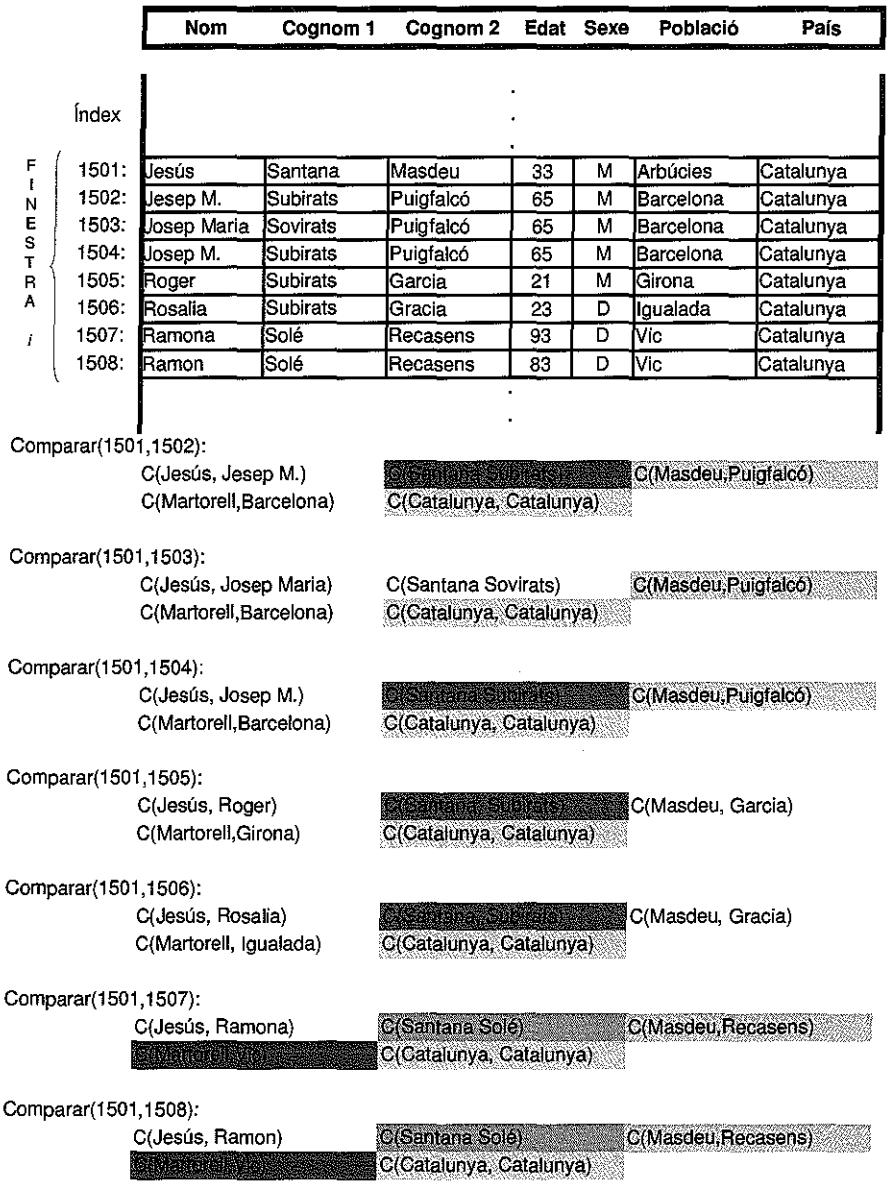

Figura 3.1: Comparacions que s'efectuen dins de la finestra i-èssima. C(x_i,y_i) cal entendre-ho com la funció que compara l'atribut i-èssim del registre x amb l'atribut i-èssim del registre y.

Per fer una justificació formal, de l'efectivitat d'emprar tècniques de memoització cal començar per entendre les característiques de les cadenes de caràcters donat que tenen unes característiques especials respecte als valor numèrics. Les cadenes de caràcters segueixen una distribució asimètrica emfatitzada pels errors en les dades.

Donat un atribut, el nombre de cadenes de caràcters diferents que pot tenir segueix la *llei de Heap* [3] en la que el vocabulari, V, té $O(N^{\beta})$ valors diferents, on N és el nombre de registres i β és un valor en l'interval $0.4 < \beta < 0.6$. La freqüència en que apareixen aquests

valors sovint compleix la *llei de Zipf* [3]. Amb una certa asimetria, la freqüència de l'i-èssim valor més freqüent és $N/(i \cdot log(N))$, tot i que s'espera una desviació més gran en dades del món real. Això vol dir que la majoria de les cadenes de caràcters apareixen pocs cops, tot i que n'hi ha un petit grup que apareix molts cops. En atributs com Nom o Cognom, per exemple, hi ha noms com Jordi, o cognoms com Garcia que són molt més freqüents que la resta. En l'Annex A es pot veure la freqüència dels noms i cognoms de Catalunya de la que disposa l'Idescat(Institut d'Estadística de Catalunya). D'aquesta observació se'n poden extreure dues conclusions:

- El nombre de valors diferents en els atributs, com Nom i Cognom, creix molt més lentament que el que creix el nombre de registres. Conseqüentment el nombre de comparacions entre strings diferents és molt més petit que el nombre de comparacions entre registres.
- Un petita fracció de l'espai de comparacions es realitza molts cops, per tant existeixen càlculs redundants en el procés.

Per tant, mentre que els mètodes de blocking tenen l'objectiu de reduir el nombre de comparacions entre registres, la memoització té l'objectiu de reduir el nombre de comparacions entre els valors dels registres que són cadenes de caràcters evitant repetir aquelles comparacions que s'han dut a terme en algun moment del procés. La tècnica de memoització en el procés de Record Linkage es va proposar en [12].

Així doncs, cal generar les estructures que permetin guardar els resultats dels càlculs que s'han anat efectuant. En el decurs d'aquests capítol es podrà veure l'estructura que guardarà la informació i com el fet d'introduir-la en el procés de Record Linkage influeix en tot el procés.

3.2 Comparison Store

L'estructura que s'encarrega de mantenir el resultat de les comparacions ja efectuades s'anomena **Comparison Store(CS)**. És una matriu triangular com la que es mostra en la Figura 3.2. Hi pot haver una CS per cada atribut. Cada parella de strings té associada una posició en la CS. Així doncs, la primera qüestió a resoldre és: com s'associa cada parella strings a una posició de la CS? De moment se suposarà que hi ha la manera per fer-ho. Més endavant s'estudiarà aquesta qüestió.

Cal tenir en compte que la mida del vocabulari farà imposible guardar en la memòria el resultat de les comparacions de totes les possibles parelles. El nombre de comparacions creix de manera quadràtica amb el nombre de cadenes de caràcters diferents. Cal doncs, guardar el resultat, W, de les comparacions entre un subconjunt de strings. El més intel·ligent és que aquest subconjunt correspongui als valors més freqüents, ja que són els que més cops s'hauran de comparar entre si. La CS, per tant, emmagatzemarà el resultat de les comparacions entre les d cadenes de caràcters més freqüents, on d depèn de la memòria disponible. Això farà que decreixin molt els requeriments d'espai en memòria per les CS, mentre que reduirà lleugerament les probabilitats d'emmagatzemar el resultat de la comparació d'una determinada parella de strings.

Vista l'estructura que emmagatzemarà el resultats, cal veure com influeix en el procés global del Record Linkage. S'ha vist que cal poder associar cada parella de valors, d'entre els més freqüents, amb una posició en la CS, que cal reservar memòria per les estructures i finalment, està clar, que les CS incidiran directament en el procés de comparació de registres. Les següents seccions del capítol descriuran en detall aquests processos.

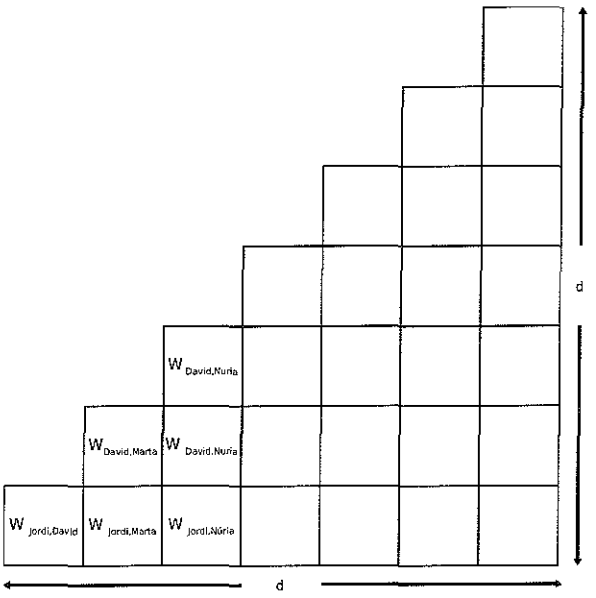

Figura 3.2: Comparison Store de l'atribut Nom

3.3 Preprocés - Model basat en Diccionaris

En aquesta secció s'explicarà com es fa per poder associar cada parella de strings, d'entre els strings més freqüents, amb una posició de la CS. Aquesta fase es considerarà el preprocés de la memoització.

La idea és crear un Diccionari per a cada atribut. Un Diccionari és una estructura de tantes posicions com valors diferents hi ha en l'atribut. Per exemple el Diccionari de l'atribut Nom de la taula de la Figura 1.1 tindria mida vuit, mentre que el de l'atribut Població tindria mida cinc, o el de l'atribut País, que tindria mida 1. En cada posició es guarda el valor i el nombre d'ocurrències d'aquest valor.

Tal i com es pot veure en la Figura 3.3, en la primera etapa d'aquesta fase, anomenada **Labeling**, es creen els Diccionaris. Com s'ha explicat, el Record Linkage inicialment llegeix els registres des d'un fitxer de registres i els carrega en la memòria. A mesura que es vagi llegint els registres, atribut per atribut, s'aniran omplint els Diccionaris. Així doncs per cada valor que es llegeixi es comprovarà si el valor existeix en el Diccionari corresponent a l'atribut que pertoqui. Si existeix, s'incrementarà el número d'ocurrències del valor en qüestió. Si no existeix, s'afegirà una nova entrada en el Diccionari amb el valor en qüestió i amb una ocurrència. En qualsevol cas, el valor, que és una cadena de caràcters, es substituirà per la posició que ocupa en el Diccionari, que és un nombre enter. La posició que el valor ocupa dins del Diccionari és l'**identificador**.

Així doncs, donats dos valors que pertanyen al mateix atribut els hi correspondrà un identificador a cadascun. A partir d'aquests dos identificadors, es podrà indexar la CS.

Com ja s'ha explicat, resulta inviable emmagatzemar els resultats de les comparacions entre totes les parelles de strings. Es vol guardar, per tant, només els resultats de les comparacions entre les d parelles de strings més freqüents. Com que s'està usant l'identificador de les cadenes de caràcters, que és la posició del valor dins del Diccionari, per indexar la CS, vol dir que només els d primers valors del Diccionari estaran representats en la CS. Cal, doncs,

una segona etapa anomenada **Translating** que ordeni el Diccionari decreixentment en funció del nombre d'ocurrències de manera que al final de la ordenació les d cadenes de caràcters més freqüents ocupin les d primers posicions del Diccionari i per tant estiguin representades en la CS. Tal i com es pot veure en la Figura 3.3, arran de la ordenació els valors han canviat de posició en el Diccionari i cal, per tant, substituir en els registres el vells identificadors pels nous identificadors que han aparegut.

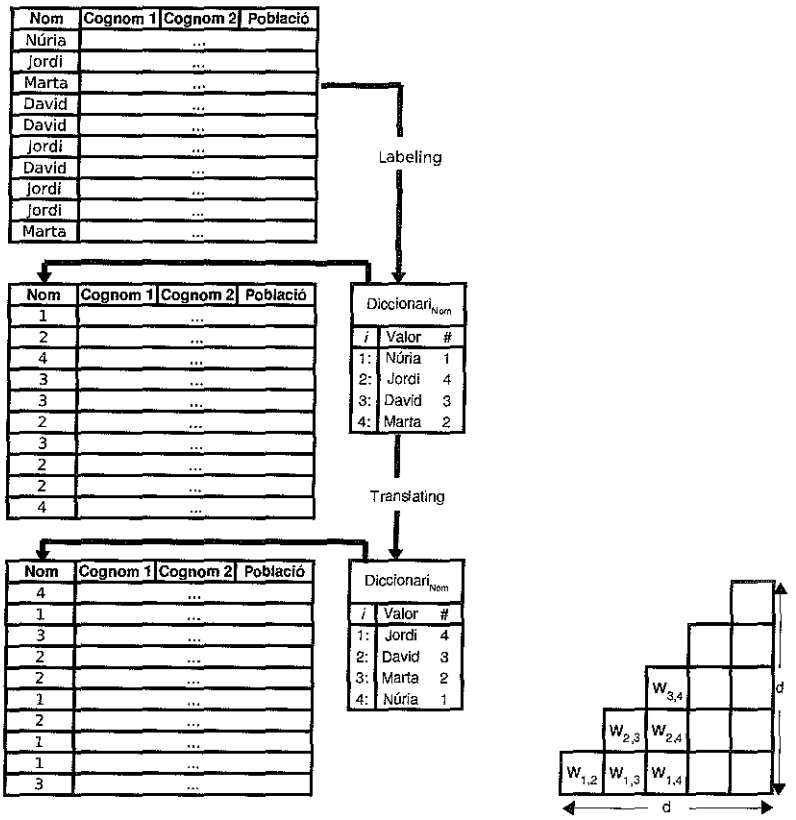

Figura 3.3: Aplicació del les etapes de Labeling i Translating sobre l'atribut Nom d'un conjunt de registres. Cada posició de la CS està indexada pels identificadors dels strings dels que guarda el resultat de la comparació.

A part d'indexar la CS amb els identificadors dels strings, la introducció d'aquesta estructura ofereix un seguit d'avantatges que la fan molt interessant:

- Un sistema d'emmagatzemament més compacte ja que s'han substituït les cadenes de caràcters per identificadors numèrics i les cadenes de caràcters ocupen més en memòria que els nombres enters.
- Es poden mantenir estadístiques sobre el nombre d'ocurrències d'un determinat valor.
- Les comparacions entre strings idèntics es poden fer de manera molt més eficient a partir dels seus identificadors, ja que la comparació de strings és molt més complexa, computacionalment parlant, que la de valors numèrics.

3.4 Reserva de memòria

S'ha dit que existeix una CS per a cada atribut els valors del qual són cadenes de caràcters. No cal oblidar que l'objectiu d'aquestes CS és minimitzar el nombre total de comparacions que cal calcular. Un cop recordat quin és l'objectiu cal també recordar la restricció que imposa el sistema, la memòria. El sistema té una determinada memòria disponible, MEM, cal doncs que la suma del que ocupa cada CS sigui com molt MEM. Si es formalitza aquest fet es dirà que, la CS_i, que correspon a l'i-èssim atribut ocupa $|D_i|$ unitats de memòria. Per tant $\sum_{i=1}^{n} |D_i| \leq MEM$.

El mètode proposa un algoritme incremental de memòria. Es van efectuant diverses iteracions, a cada iteració s'escull una CS que té la **densitat** més alta. La densitat és la fracció entre la mida de la CS i la quantitat de valors representats en la CS. La CS escollida veu augmentat en una unitat el seu nombre de valors representats i la seva mida es veu incrementada d_k unitats. Es surt del bucle quan cap de les CS pot incrementar la seva mida sense superar la memòria disponible, MEM, o bé totes les CS estan representant tots els valors possibles de l'atribut.

És interessant observar com l'objectiu, explicat d'una manera més informal, és utilitzar el màxim de la memòria disponible del sistema entre les diferents CS, i a més fer-ho de manera que se'n maximitzi el rendiment, és a dir, que els valors més freqüents de tots els atributs amb CS estiguin emmagatzemats en la CS corresponent.

3.5 Comparació de Registres

És interessant confrontar la Comparació de Registres, quan no s'aplica memoització amb quan se'n aplica. Per fer-ho es suposarà que s'han de comparar els dos registres de la Figura 3.4.

Josep M.	Subirats	Puigfalcó	Barcelona	Catalunya
Rosalia	Silvestre	Garcia	Igualada	Catalunya

Figura 3.4: Dos registres.

Comparar-los, sense aplicar memoització, voldrà dir, per a cada atribut, comparar els corresponents valors: Josep M. amb Rosalia, Subirats amb Silvestre, Puigfalcó amb Garcia, Barcelona amb Igualada i Catalunya amb Catalunya, en total cinc comparacions. Un cop fet això es podrà computar el grau de similitud global entre els dos registres.

Ara es suposarà, tenint en compte que s'aplica memoització, els Diccionaris i les CS que es poden veure, de manera parcial, en la Figura 3.5.

Cal tenir en compte que un cop passades les etapes de Labeling i Translating els registres no estan composats per cadenes de caràcters, ho estant pels seus identificadors. En la Figura 3.5 es pot veure l'identificador pel que s'ha substituït cada string. Comparar els dos registres voldrà dir comparar 23 amb 55 per l'atribut Nom, 66 amb 98 per l'atribut Cognom 1, 2 amb 74 per l'atribut Cognom 2, 3 amb 55 per l'atribut Ciutat i 5 amb 5 per l'atribut País. Per comparar 23 amb 55, es va a la posició (23,55) de la CS de l'atribut Nom i es veu que tenen una similitud d'un 0'1. El procés és el mateix pels atributs Cognom 1 i Cognom 2. A l'hora de comparar el valor de l'atribut Ciutat, és a dir, 3 amb 55, es va a la posició (3,55) de la CS de l'atribut Ciutat i es veu que aquest valor no ha estat calculat mai. Cal,

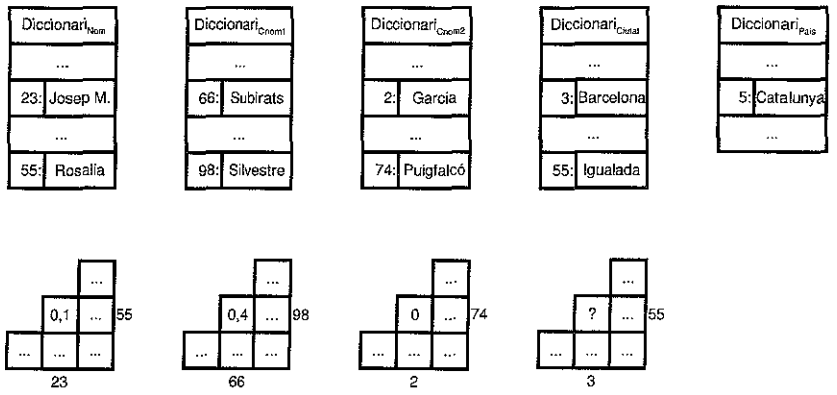

Figura 3.5: Diccionaris parcials, sense mostrar el nombre d'ocurrències, i CS parcials que mostren només aquell parell de registres que es poden veure en el corresponent Diccionari.

doncs, recuperar els valors reals mitjançant els Diccionaris: Barcelona i Igualada i comparar-los mitjançant la funció de comparació de strings. Un cop el càlcul està realitzat, es va a la posició (3,55) de la CS de l'atribut Ciutat i s'hi guarda el valor calculat. Ja no caldrà calcular mai més, en aquest procés, aquesta comparació. A l'hora de comparar els valors de l'atribut País, cal comparar 5 amb 5, però al veure que els dos valors tenen el mateix identificador es pot respondre directament que els dos valors són idèntics. Notis que quan no s'utilitza memoització s'han hagut de fer cinc comparacions entre cadenes de caràcters, mentre que quan si que se'n utilitza ha calgut efectuar-ne només una, corresponent a la de l'atribut Ciutat.

A partir d'aquest moment es distingirà el concepte de **comparació real**, que farà referència a quan dos strings s'han de comparar mitjançant la funció de comparació, del concepte **falses comparacions**, que farà referència a quan la comparació real s'ha efectuat en algun moment i només cal anar a buscar el resultat d'aquesta comparació en la CS.

Un cop vist que la memoització redueix el número de comparacions entre atributs perquè hi ha càlculs que ja ha efectuat en algun moment i se n'ha guardat el resultat, és interessant veure una altra possibilitat que ofereix el sistema i que redueix encara més el número de comparacions.

Com ja s'ha explicat només aquelles parelles de registres que tinguin un grau de similitud W, que superi un llindar mínim T, són marcats com a registres similars. Per calcular el grau de similitud global es calculaven els graus de similitud parcials, atribut a atribut. Així doncs, donats dos registres x i y compostos per n atributs, W es calcula així: $(W(x,y) = \sum_{i=1}^{n} W_i(x_i, y_i)$. Si es suposa aquest procés, com un procés seqüencial de n passos, a cadascun d'ells es pot comprovar si encara és possible, o no, arribar al llindar mínim T. És a dir, abans d'efectuar la j-èssima comparació: $(\sum_{i=1}^{j-1} W_i) + (\sum_{i=j}^{n} ACCEPTACIO_i) < T$, on $\sum_{i=j}^{n} ACCEPTACIO_i$ és la suma dels graus de similitud parcials que encara no s'han calculat suposat que aquests graus són màxims. Si la inequació es compleix vol dir que no cal seguir efectuant comparacions entre els atributs dels registres ja que mai s'arribarà a assolir el llindar mínim, es produïra una **interrupció prematura** de la comparació. Si la inequació no és compleix cal seguir fent les comparacions per, finalment, obtenir quin és el grau global de similitud entre els dos registres[1].

[1]Com ja s'ha explicat el procés de Record Linkage acaba amb la revisió per part d'un expert que és qui

Això obre una nova possibilitat en la comparació de registres. Donats dos registres, per a cada comparació entre els seus atributs es comprova si el resultat es troba, o no, en la CS. Si el resultat no s'hi troba, aleshores, es posposarà la comparació. Si en algun moment es produeix una interrupció prematura de la comparació, aquelles comparacions que havien estat posposades no caldrà efectuar-les.

Cal deixar palès que per comparar els registres es poden seguir aplicant els mètodes de blocking estudiats anteriorment. Per realitzar els blocs, o les diferents ordenacions, ara, però, caldrà anar a buscar el valor dins del Diccionari, ja que els registres només contindran valors numèrics. Deixant de banda aquest fet, el procés resulta exactament el mateix, i per tant la seva aplicació continua sent perfectament vàlida.

realment decideix si dos registres similars, segons els procés de RL, corresponen, o no, a la mateixa entitat real. Tenir, per tant, la similitud global entre els dos registres pot ajudar a prendre la decisió.

3.6 Conclusions de les tècniques de memoització

És interessant, arribat a aquest punt, fer una parada i veure en que s'ha convertit el procés de Record Linkage. En la Figura 3.6 es pot veure un esquema complert.

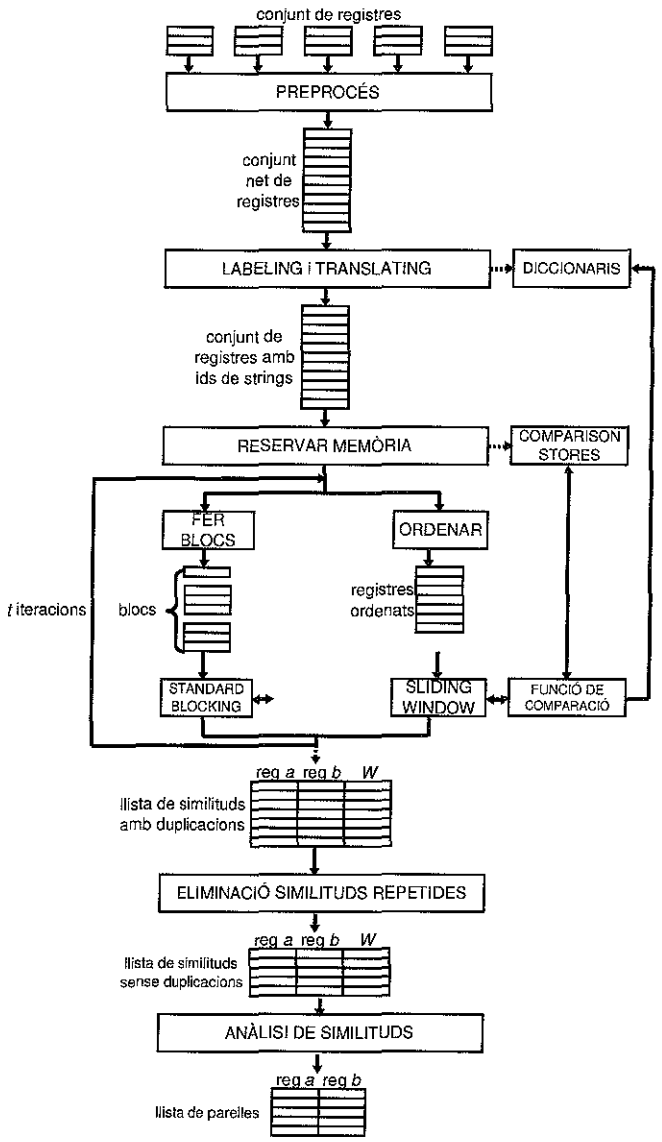

Figura 3.6: Esquema complert del procés de RL aplicant tècniques de blocking i de memoització. Notis que malgrat no està dibuixat, en cas de seguir el mètode de Standard Blocking, també es fa servir la funció de comparació, així com, les CS i els Diccionaris.

Es pot observar en la Figura 3.6, que s'han recuperat les dues etapes del RL que s'havien ignorat fins el moment: Preprocés i Anàlisi de similituds. És interessant recordar-les per no oblidar que RL és un procés que comença amb una fusió de diferents fitxers de registres i acaba amb l'anàlisi de les similituds per part d'un expert, que és qui finalament decideix si dos registres corresponen a la mateixa entitat. A partir d'aquest moment, si no es diu el contrari, quan es faci referència al procés de RL es seguirà ignorant aquestes dues etapes per facilitar la lectura. Un cop es tenen els diferents registres fusionats, es du a terme el

preprocés corresponent a la memoització, és a dir, el Labeling i el Translating, d'on sorgeixen els Diccionaris. A continuació es reserva, mitjançant algoritmes d'optimització, la memòria destinada a les diferents CS. Seguidament, amb el registres, en els que s'han substituït els strings per identificadors, es procedeix a realitzar els blocs o la ordenació, sobre els que s'aplica Standard Blocking o Sliding Window respectivament. Això es repeteix durant t iteracions, donant com a resultat una llista de similituds de la que que cal eliminar aquelles parelles de similituds que s'han trobat diferents vegades en les diferents iteracions. Finalment, i com ja s'ha dit, és un expert el que s'encarrega de decidir si les similituds trobades corresponen, o no, a la mateixa entitat real.

Cal destacar que el fet d'emprar memoització en el procés de RL no afecta per res el recall ni la precisió de les tècniques de blocking. I per altra banda, cal recordar que l'aplicació de memoització no afecta absolutament per res la complexitat del procés que segueix depenent dels N registres i sent $O(B' \cdot N)$, on B' és la mida del bloc més gran en el cas d'emprar Standard Blocking, $O(w \cdot N)$, on w és la mida de la finestra en el cas d'emprar Sliding Window o $O(N^2)$, en el cas que no s'apliqui cap mètode de blocking. El que s'aconsegueix amb la tècnica de memoització és estalviar-se temps de càlcul evitant repetir les mateixes comparacions.

En Figura 3.7 es pot veure el temps mitjà per comparació de registres que s'ha obtingut combinant memoització amb diferents mètodes de blocking. Les configuracions del mètodes de blocking que s'empren són les que es poden veure en la Taula 2.1. La màquina amb la que s'ha executat l'experiment és un 2 dual-core a 1,65GHz amb 4GB de memòria. S'observa que pel fet d'introduir la CS els temps han disminuït, en el pitjor dels casos fins gairebé la meitat (B-A 10K), mentre que en el millor dels casos el temps s'ha vist reduït fins a, gairebé, cinc vegades (B-A 1000K). Aquests temps mostren que l'aplicació de les tècniques de memoització resulten molt efectives en el procés de RL.

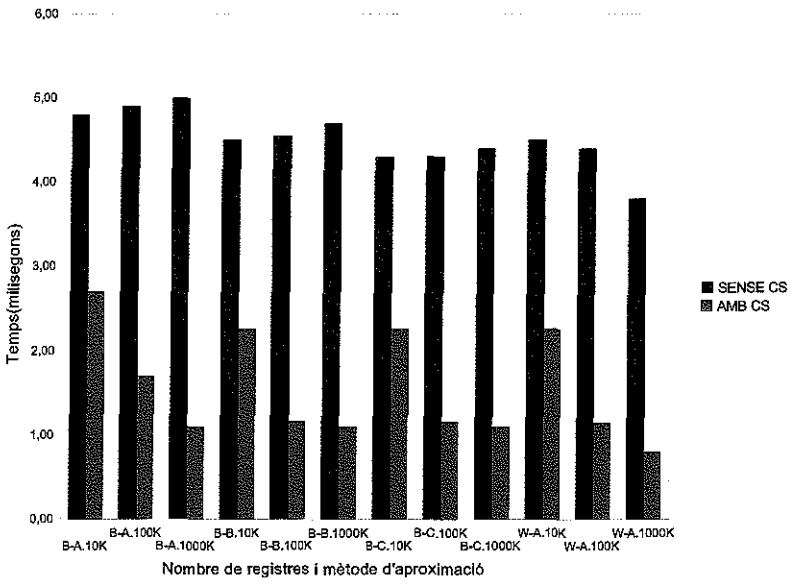

Figura 3.7: Temps mitjà de comparació per comparació de registres amb diferents mètodes de blocking i configuració d'aquest, utilitzant, i no, memoització.

S'ha vist fins el moment que el procés de Record Linkage s'ha pogut accelerar mitjançant

els mètodes de blocking i l'aplicació de les tècniques de memoització. Tot i això, el procés de Record Linkage es seguix considerant un procés lent, i cal, per tant, buscar d'altres maneres que permetin accelerar-lo. Explotades les vies més convencionals per accelerar el procés de RL cal començar a plantejar-se d'altres vies. La paral·lelització sembla ser una bona idea ja que ha de permetre a aquelles institucions que han d'efectuar el procés de Record Linkage, però no d'altres càlculs complexos, que el puguin fer de manera ràpida sense haver d'invertir en nou maquinari.

Capítol 4

Record Linkage paral·lel

En aquest capítol es procedirà a explicar la paral·lelització distribuïda del procés de Record Linkage. En primer lloc, s'identificarà quina és la feina paral·lelitzable i com cal dividir-la per mantenir bons recalls i bona precisió. Seguidament, s'explicarà com aquesta divisió de la feina afecta al conjunt de tot el procés de Record Linkage, això serà la **paral·lelització base** del procés de Record Linkage. A partir d'aquesta paral·lelització es proposaran diferents modificacions. Un primer bloc d'aquestes modificacions, explicades en les Seccions 4.4 , 4.5 i 4.6, tenen com a objectiu realitzar el procés de Record Linkage el més ràpid possible. En la Secció 4.7 es presenta una modificació que canvia l'aproximació a la paral·lelització. Si fins ara l'objectiu principal havia estat efectuar el procés de Record Linkage tan ràpid com fos possible, aquesta modificació té com a objectiu treballar amb quantitats més grans de registres, encara que es perdi una mica de velocitat.

Es paral·lelitzarà un software ja existent, desenvolupat per DAMA-UPC, anomenat DAURUM. Aquest software executa un procés de Record Linkage aplicant la tècnica de memoització descrita en el Capítol 3 i el mètodes de blocking descrits en el Capítol 2.

4.1 Divisió de la feina

En la Secció 1.2 s'ha dit que la paral·lelització consisteix en, donada una gran tasca dividir-la en d'altres de més petites. Així doncs, sembla quelcom natural començar identificant la tasca concreta del procés de Record Linkage que es pot dividir.

La tasca més costosa era, i segueix sent, l'etapa de Comparació de Registres (veure Figura 2.1), així doncs, és aquesta tasca la que cal veure com es paral·lelitza. Dins d'aquesta tasca, i com ja s'ha explicat, els mètodes de blocking divideixen el conjunt de registres en blocs i treballen amb cada bloc de manera independent. Si es treballa amb cada bloc de manera independent vol dir que es pot treballar amb els diferents blocs de manera concurrent, en paral·lel.

Per una banda hi ha Standard Blocking que divideix els registres en blocs, la mida dels quals depèn de la clau de bloc que s'hagi escollit i a més no tenen perquè tenir la mateixa mida entre ells, de fet és quelcom poc probable. Per altra banda hi ha Sliding Window, que donat un conjunt de registres treballa amb blocs, finestres, que són tots de la mateixa mida, ja que és un paràmetre configurable.

Una màxima en el món de la computació paral·lela és el **balanceig de càrrega**. El balanceig de càrrega consisteix en dividir el volum de feina de manera que tots els processadors que l'han de realitzar en tinguin la mateixa quantitat, d'aquesta manera es garanteix un millor rendiment.

Donades aquestes premises sembla que el que pot resultar més senzill, i a l'hora més eficient, és paral·lelitzar el mètode de Sliding Window. Això és, donat el conjunt de registres dividir-lo en S conjunts de registres, on S és el nombre de processadors que realitzaran la tasca. A partir d'aquest moment cadascun d'aquests conjunts rebrà el nom de **Sfinestra**. Cal veure com es creen aquestes Sfinestres per no perdre el recall que el mètode de Sliding Window assoleix. Per garantir-lo cal que un determinat registre es compari exactament amb els mateixos registres en la versió seqüencial que en la paral·lela.

La manera més naïf de crear les Sfinestres és fent una divisió en S Sfinestres independents de manera que cada processador tingui una Sfinestra de N/S registres, on N és el nombre total de registres i S el de processadors. És a dir, si hi ha N registres, $r_1, r_2, r_3, \cdots, r_N$ i S processadors, cada Sfinestra, SF_i, és conforma així: $SF_0 = \{r_1, \cdots, r_{\frac{N}{S}}\}$, $SF_1 = \{r_{\frac{N}{S}+1}, \cdots, r_{\frac{2N}{S}}\}, \cdots$ $SF_{S-1} = \{r_{\frac{(S-1)N}{S}+1}, \cdots, r_{\frac{SN}{S}}\}$. A partir de les Taules 4.1 i 4.2 es pot veure que aquesta divisió difícilment assolirà el recall i la precisió assolits pel mètode de Sliding Window. En la primera taula es pot veure les comparacions que es produeixen aplicant el mètode de Sliding Window de manera seqüencial, és a dir sense haver de dividir el conjunt de registres, a un conjunt de setze registres, i tenint en compte que la mida de la finestra és de quatre registres. Les files (x) i les columnes (y) representen registres, si en una posició (x, y) hi ha una 'X' significa que es produeix una comparació entre el registre x i el registre y.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		X	X	X												
2			X	X	X											
3				X	X	X										
4					X	X	X									
5						X	X	X								
6							X	X	X							
7								X	X	X						
8									X	X	X					
9										X	X	X				
10											X	X	X			
11												X	X	X		
12													X	X	X	
13														X	X	X
14															X	X
15																X

Taula 4.1: Comparacions que es produeixen entre un conjunt de setze registres suposant el mètode de Sliding Window amb finestra de mida quatre.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		X	X	X												
2			X	X	X											
3				X	X	X										
4					X	X	X									
5						X	X	X								
6							X	X								
7								X								
8																
9										X	X	X				
10											X	X	X			
11												X	X	X		
12													X	X	X	
13														X	X	X
14															X	X
15																X

Taula 4.2: Comparacions que es produeixen entre un conjunt de setze registres suposant el mètode de Sliding Window amb finestra de mida quatre i suposant dues Sfinestres.

Si a partir del conjunt de registres de la Taula 4.1, suposant dos processadors, es creen

les Sfinestres de la manera que s'ha descrit, es pot veure, en la Taula 4.2, que s'han deixat de fer algunes comparacions entre registres, i per tant s'està donant la possibilitat de perdre recall. De fet s'observa que les comparacions que falten són: registre 6 amb el registre 9, registre 7 amb els registres 8 i 9, i registre 8 amb els registres 9, 10 i 11. Això és degut a que la divisió del conjunt de registres en dues Sfinestres ha eliminat les comparacions que es feien en les finestres que queien entre les dues Sfinestres. En aquest cas les finestres que van del registre 6 al 10, del 7 a l'11 i del 8 al 12. Cal que aquestes comparacions es tornin a fer, la millor manera és definint la segona Sfinestra del registre 6 al 15, en comptes de definir-la del registre 9 al 15.

Si es generalitza aquesta observació es pot concloure que donat un conjunt de registres dividits en S Sfinestres, $SF_0, SF_1, ...SF_{S-1}$, per mantenir el recall, cal que, per cada Sfinestra SF_i, per a $0 < i < S$ els primers $w-1$ registres, on w és el nombre de registres que conformen una finestra, siguin els mateixos que els últims $w-1$ registres de SF_{i-1}. És a dir, que a partir de SF_1 existeix un solapament de $w-1$ registres amb la Sfinestra immediatament anterior. Així doncs, la mida de SF_0 serà de N/S registres, la mida de SF_i per a $0 < i < S-1$ serà $N/S+(w-1)$ registres i finalment la mida de SF_{S-1} serà $N/S+(w-1)+r$, on r és el residu de la divisió N/S.

En aquesta secció s'ha detectat quina feina era divisible en d'altres més petites i s'ha vist com fer aquesta divisió per garantir el recall que oferia la versió seqüencial del procés de Record Linkage. En les següents seccions es veurà com es produeix aquesta divisió de la tasca, és a dir qui se n'encarrega, i com afecta a tot el procés.

4.2 Paradigma Mestre-Esclau

La paral·lelització proposada segueix el paradigma Mestre-Esclau. En aquest paradigma existeix un node, el Mestre, que executa essencialment les parts seqüencials del programa i distribueix la feina paral·lelitzable entre els diferents nodes que estan desocupats, els Esclaus. Quan un Esclau finalitza la seva feina informa al Mestre i aquest li assigna una nova quantitat de feina. És a dir, en les paral·lelitzacions que segueixen aquest paradigma hi ha un sol coordinador que és el que té una visió global del procés i que és conscient de la feina que s'ha de fer, i per tant és l'encarregat de repartir aquesta feina entre els altres nodes.

Cal veure, doncs, com s'aplica aquest paradigma a la paral·lelització del procés de Record Linkage.

En la Figura 4.1 es pot veure l'esquema que es seguirà. Es **suposarà que el conjunt de registres està emmagatzemat en un sol node**, el Mestre. El Mestre ordenarà, a cada iteració del procés, el conjunt de registres per la clau d'ordenació que toqui, cal recordar que s'està paral·lelitzant un procés de Record Linkage al que s'aplica el mètode de Sliding Window, i que per tal d'augmentar el recall d'aquest s'efectuen t iteracions amb diferents claus d'ordenació.

Un cop el conjunt de registres estigui ordenat el dividirà, tal i com s'ha explicat en la secció anterior, en S Sfinestres, on S és el nombre d'Esclaus que hi ha. Un cop hagi creat les Sfinestres enviarà cada Sfinestra a un Esclau diferent. Per fer-ho, caldrà que construeixi un missatge MPI amb la informació. En el moment immediat que l'Esclau rebi una Sfinestra es posarà a executar Sliding Window sobre aquesta. Quan hagi acabat d'efectuar el procés enviarà un missatge al Mestre informant-lo que està disponible per rebre una Sfinestra de la següent iteració. Notis que en tot el procés el Mestre sempre té tot el conjunt de registres, la Sfinestra que envia a cada Esclau és un conjunt de registres format per còpies dels registres originals.

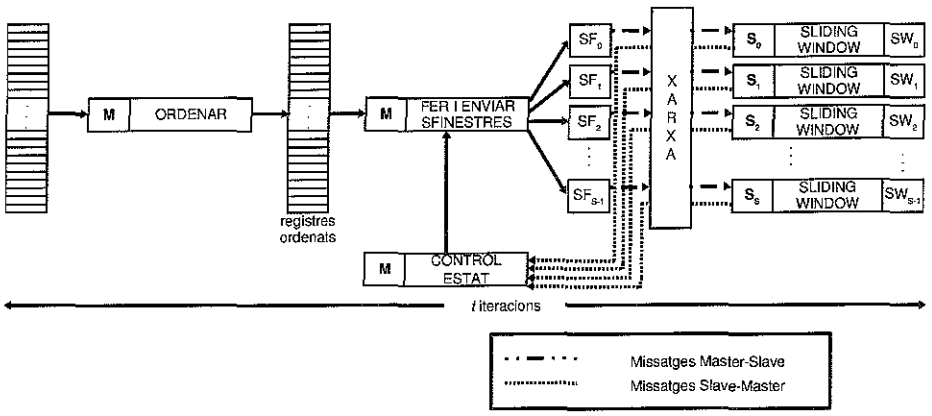

Figura 4.1: Paral·lelització de l'etapa de Comparació de Registres del procés de Record Linkage seguint un model Mestre(M)-Esclau(S_i).

4.3 Adaptació del procés de Record Linkage a la paral·lelització: Paral·lelització base

Seguir el model Mestre-Esclau tal i com s'ha descrit en la secció anterior comporta una sèrie d'implicacions en el conjunt de les etapes que conformen el procés de Record Linkage que aplica la tècnica de memoització que s'ha vist en el Capítol 3. A continuació es descriuran aquestes implicacions i es veuran els canvis que cal dur a terme en el procés perquè tingui un funcionament correcte.

En l'execució seqüencial. Preprocés de la memoització:

Com ja s'ha explicat, els registres, quan s'han de comparar entre si, no contenen strings. Tots els valors que eren d'aquest tipus han estat substituïts per identificadors de strings, que són nombres enters. Els identificadors s'utilitzen per indexar la **Comparison Store** (CS) corresponent a l'atribut que s'està comparant. Si la CS té emmagatzemat el resultat de la comparació no cal fer res més. Sinó, cal anar al **Diccionari**, també corresponent a l'atribut que s'està comparant, recuperar els valors originals (les cadenes de caràcters), realitzar la comparació mitjançant una funció de comparació de caràcters i guardar el resultat en la corresponent posició de la CS.

En l'execució paral·lela. Preprocés de la memoització:

S'observa que en la fase de comparació de registres intervenen dos tipus d'estructures: la Comparison Store i el Diccionari. Per tant, cal que els Esclaus, que són els encarregats comparar els registres, aplicant Sliding Window, disposin d'aquestes estructures per poder realitzar les comparacions.

Tal i com s'ha definit en la Secció 3.3 un Diccionari conté tots els valors diferents que hi ha i el nombre de vegades que apareix cadascun en l'atribut al que pertany. Així doncs, per crear els Diccionaris és necessari disposar de tots els registres que intervindran en el procés de Record Linkage. Tal i com s'ha plantejat la paral·lelització, el Mestre és l'únic que té tot el conjunt de registres, per tant és l'únic que és capaç, mitjançant les etapes de Labeling i Translating, de crear els Diccionaris. Serà, doncs, el Mestre qui els creï, però com que els Esclaus els necessitaran per efectuar la comparació de registres, el Mestre els hi haurà d'enviar en forma de missatge.

L'altra estructura que s'ha detectat que els Esclaus han de menester per dur a terme la fase en la que es comparen els registres és la Comparison Store. Per crear les Comparison Stores cal calcular la mida que han de tenir en funció del nombre de valors diferents que hi ha en l'atribut al que pertanyen i de la memòria disponible. La informació del nombre de valors diferents que hi ha en l'atribut es pot saber molt fàcilment a partir del Diccionari de l'atribut. Com que els Diccionaris s'han hagut d'enviar no resulta cap problema, ara, que cada Esclau creï les seves pròpies CS.

En l'execució seqüencial. Eliminació similituds duplicades:

Per veure en quin altre punt cal adaptar el procés pel seu funcionament paral·lel cal recordar una altra fase necessària que forma part del procés de Record Linkage al que s'aplica el mètode de Sliding Window. Com ja s'ha explicat en la Secció 2.3 per augmentar el recall del mètode s'efectuen diferents iteracions emprant diferents claus d'ordenació. Aquest fet té com a conseqüència que una mateixa similitud es pot trobar més d'un cop. Per pal·liar aquest efecte s'introduïa una nova etapa d'eliminació de similituds repetides.

En l'execució paral·lela Eliminació similituds duplicades:

En el procés de Record Linkage paral·lel s'ha definit que cada Esclau du a terme la fase de Sliding Window sobre les seves Sfinestres, això vol dir que al final del procés cada Esclau té una llista de similituds. Un mateix Esclau pot tenir diverses similituds repetides, però també poden estar repetides les similituds entre els diferents Esclaus. Així doncs, per dur a terme l'etapa d'eliminació de similituds repetides cal que els Esclaus enviïn les similituds que han trobat al Mestre. El Mestre, un cop les hagi rebut totes, pot eliminar, de la mateixa manera que feia fins ara, les similituds repetides.

En la Figura 4.2 es pot veure un esquema complert de la paral·lelització base.

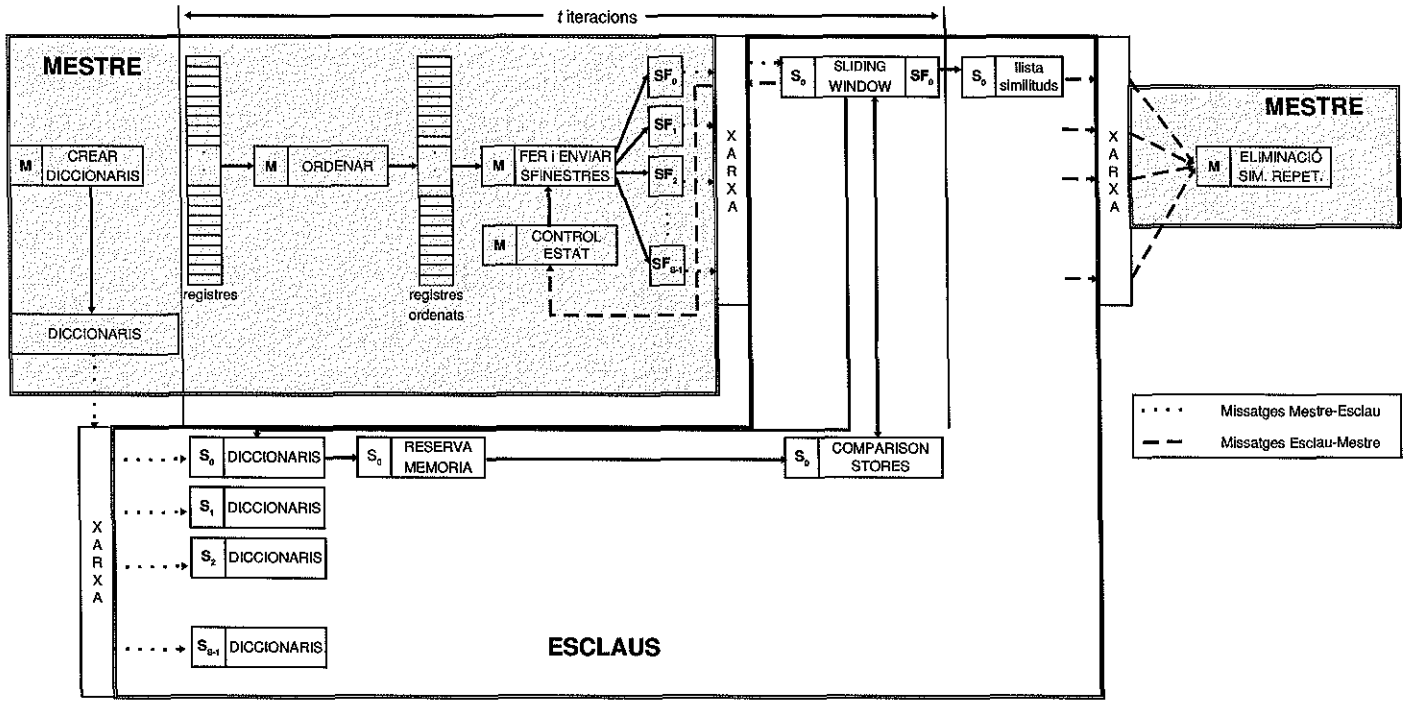

Figura 4.2: Esquema del procés base de Record Linkage paral·lel amb un Mestre(M) i S Esclaus(S_i). Per simplificar la imatge es mostra només el comportament de l'Esclau 0. S'observa com el Mestre crea els Diccionaris i els envia a través de la xarxa als diferents Esclaus. Cada Esclau a partir dels Diccionaris reserva la memòria i crea la seva pròpia col·lecció de Comparison Stores. Un cop el Mestre ha enviat els Diccionaris ordena el conjunt de registres per la primera clau d'ordenació i a partir d'aquesta ordenació crea les Sfinestres que envia a cada Esclau. Un cop els Esclaus reben les Sfinestres comencen a fer la comparació dels registres emprant el mètode de Sliding Window. Per fer les comparacions cada Esclau fa ús de la seva col·lecció de Comparison Stores i de Diccionaris. Quan l'Esclau acaba de buscar similituds sobre una Sfinestra avisa al Mestre perquè aquest li pugui enviar una Sfinestra de la següent iteració. Tot aquest procés, d'ordenar, fer Sfinestres, enviar-les i comparar registres es repeteix t vegades amb diferents claus d'ordenació. Un cop s'han fet les t iteracions cada Esclau envia la llista de similituds que ha trobat al Mestre, que un cop les ha rebudes totes elimina aquelles que estan duplicades.

Un cop vista l'adaptació de la paral·lelització seguint un model Mestre-Esclau és interessant veure, de manera explicita, com queda el procés des d'un punt de vista de responsabilitats i temporal. En la Figura 4.3 es pot veure tot el procés de Record Linkage d'una manera molt clara i entenedora i per altra banda deixa palès quines són les parts que s'executen de manera paral·lela i quines de manera seqüencial.

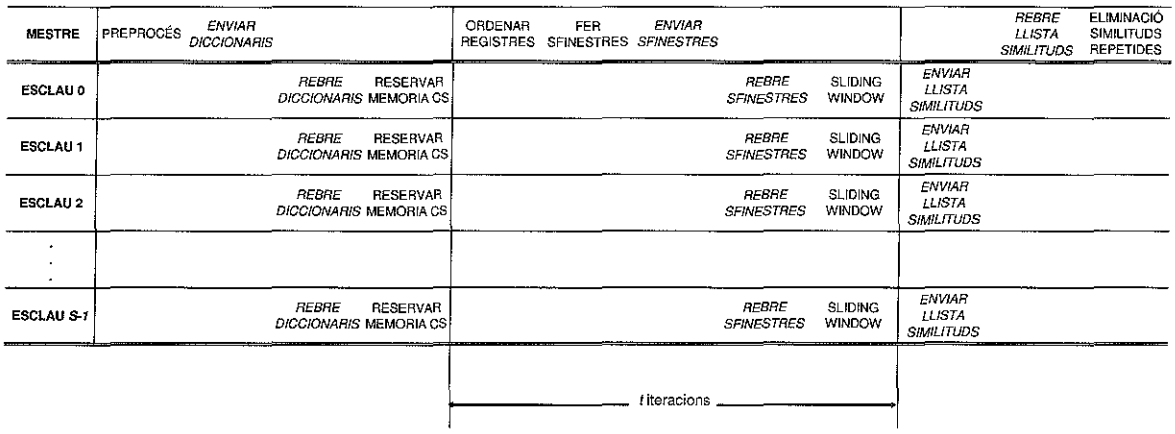

Figura 4.3: Escala temporal dividida en responsabilitats del procés de Record Linkage paral·lel que segueix un model Mestre-Esclau.

Dins de les tasques seqüencials es poden trobar el preprocés, és a dir, les etapes de Labeling i Translating que tenen com a resultat la creació dels Diccionaris, la ordenació dels registres, la divisió dels registres en Sfinestres i l'eliminació de similituds repetides, com a conseqüència de les t iteracions que s'efectuen per augmentar el recall. Notis que tot i que la ordenació dels registres i la seva divisió en Sfinestres es fa de manera seqüencial en el Mestre, mentre els Esclau executen Sliding Window, es pot començar ja a fer la ordenació i divisió dels registres de la següent iteració. Les tasques paral·leles són la creació de les Comparison Stores, i la més important, la fase de Sliding Window.

Fins el moment s'ha vist l'aproximació més senzilla a la paral·lelització del procés de Record Linkage. Aquesta aproximació, però, es pot veure modificada tenint en compte d'altres factors. En les properes seccions es podran veure les modificacions que es proposen. Val a dir que aquestes modificacions no són incrementals entre si, és a dir, cadascuna d'elles es combina sobre la versió base de la paral·lelització, però no entre elles.

4.4 Mestre-Treballador

S'ha explicat que es segueix el paradigma Mestre-Esclau. És a dir, que els que realment realitzen la feina ded procés paral·lel són els Esclaus. Mentre els Esclaus estan duent a terme la fase de Sliding Window sobre les seves Sfinestres el Mestre tan sols ha de ordenar el conjunt de registres per la següent clau d'ordenació i dividir-lo en Sfinestres per enviar als Esclaus.

Cap la possibilitat, per tant, que el node Mestre faci també d'Esclau. Això comporta que a l'hora de fer les Sfinestres hi hagi un Esclau més, pel que cada Sfinestra serà més petita i per tant el temps que triga un Esclau en efectuar les comparacions entre els registres de la Sfinestra haurà de ser menor. Es distingirà la versió en la que el Mestre no actua com a Esclau, **Mestre-Gandul**, de la versió en que actua com a Esclau, **Mestre-Treballador**.

En el Capítol 5 s'analitzaran, i compararan, en base a uns experiments ambdues versions.

4.5 Paral·lelització a nivell de node

Com ja s'ha introduït en la Secció 1.2 hi ha dos tipus d'ordinadors paral·lels. Fins ara s'ha centrat l'atenció en els sistemes paral·lels distribuïts, és a dir, sistemes que estan formats per múltiples ordinadors connectats entre ells mitjançant una xarxa. S'ha explicat la paral·lelització, i l'adaptació que ha calgut fer del procés de Record Linkage per tal de que funcionés sobre aquesta mena de sistemes. Ara bé, cal tenir en compte que cadascun d'aquests ordinadors que conformen el sistema distribuït poden ser ordinadors paral·lels de memòria compartida (**SMP**-Shared-memory MultiProcessor), és a dir ordinadors multi-core. Això fa possible paral·lelitzar a nivell de cada ordinador.

Fins el moment cada node rebia una Sfinestra i dins d'aquesta Sfinestra comparava els registres seguint el mètode de Sliding Window. Això segueix sent així, però amb la diferència que ara les comparacions entre registres, en comptes de fer-les una única CPU, les fan tantes CPUs com l'ordinador multi-core té. D'aquesta manera s'espera accelerar, encara més, el procés.

És interessant fer una petita reflexió sobre com funciona la comparació de registres i la interacció amb les CS quan es du a terme una paral·lelització a nivell de node. Com s'ha explicat, s'escriu el resultat d'una comparació en una posició de la CS quan la comparació de dos strings s'efectua per primer cop. Dit això, és interessant preguntar-se què passaria si dos, o més, processadors, en un moment donat estiguessin comparant la mateixa parella de strings, que no s'havia comparat mai abans. Tots els processadors faran la comparació entre els strings i a mesura que completin el càlcul escriuran el resultat de la comparació en la corresponent posició de la CS. Notis que com que els processadors estan comparant la mateixa parella de strings tots escriuran en la mateixa posició de la CS. Això podria fer pensar que s'està substituint els valors escrits per altres processadors i en certa manera és cert, però s'està substituint pel mateix valor. Això és degut a que la funció de comparació és **determinista**. Això permet estalviar-se tot el control sobre la CS, i simplifica el seu ús sobre la paral·lelització de memòria compartida.

En el Capítol 5 s'analitzarà com interacciona la paral·lelització de memòria distribuïda amb la paral·lelització de memòria compartida.

4.6 Compartint el contingut de les CS

Com ja s'ha vist fins el moment cada Esclau té la seva pròpia col·lecció independent de Comparison Stores. Això vol dir que en global es fan més comparacions reals, sumant les que es fan a tots els Esclaus, en la versió paral·lela del Record Linkage, que no pas en la versió seqüencial.

Aquesta afirmació cal agafar-la amb pinces ja que això serà així sempre i quan la mida de la CS sigui la mateixa en la versió paral·lela que en la versió seqüencial. Per entendre més fàcilment el raonament suposarem unes CS infinites en les que es poden guardar tots els resultats de totes les comparacions. Així doncs, donada una parella de valors qualssevol, en el cas seqüencial només es farà una comparació real entre aquest parell de valors, la resta de comparacions que s'hagin de fer es traduiran en consultes a la Comparison Store. En la paral·lelització, donats dos valors qualssevol, es poden arribar a efectuar S comparacions reals, una a cada Esclau.

L'objectiu és, doncs, reduir el nombre global de comparacions globals que s'efectuen en la paral·lelització.

Hi ha dues possibilitats a contemplar:

1 Cada cop que un Esclau realitza una comparació real, i per tant s'introdueix el resultat en la CS, enviar el resultat a tots els altres Esclaus. La resta dels Esclaus és segur que no l'han calculada abans, perquè si ho haguessin fet, haurien enviat el resultat. Aquesta aproximació aconseguirà que la paral·lelització faci exactament el mateix nombre de comparacions reals que la seqüencial. Aquesta aproximació, però, no és assumible: la quantitat de valors que s'haurien d'enviar de manera ininterrompuda a través de la xarxa acabarien per saturar-la.

2 Cada cert temps, fusionar totes les CS i enviar-les a tots els Esclaus. Aquesta aproximació farà que es redueixin el nombre global de comparacions reals en la paral·lelització tot i que seguirà sent més alt que en la versió seqüencial ja que en aquest cert temps que passa abans de que es fusionin les CS es poden haver repetit comparacions. Tot i això, aquesta segona aproximació sembla que pot donar un millor rendiment perquè fa pocs enviament de moltes dades, a diferència de l'altra opció que fa molts enviament de poques dades.

Escollida aquesta segona aproximació cal veure com es duu a terme. S'ha dit que cada cert temps es fusionaran les CS. Tot i que es podria posar un rellotge sembla més lògic buscar un moment adequat en el procés per fer aquesta fusió. Si es mira l'esquema de la Figura 4.2 es pot veure que un bon moment per dur-la a terme pot ser just en acabar una iteració i abans de començar la següent. A cada iteració les CS es van omplint, però si que és cert que en les primeres iteracions s'ompliran molt més que en les últimes. Es pot contemplar la possibilitat de fusionar les CS al finalitzar cadascuna de les iteracions d'entre les primeres t' iteracions. Per altra banda, com ja s'ha explicat en el Capítol 2 hi ha uns quants valors que es repeteixen molt més que la resta, de fet els primers valors de les CS corresponen als valors que més es repeteixen dins de l'atribut al que pertanyen. Pot ser interessant, en comptes d'enviar tot el contingut de totes les CS, enviar el subconjunt de d' elements més freqüents de cada CS.

La primera idea que hom pot tenir a l'hora de fusionar la CS és enviar totes les CS, o el subconjunt d'elements que es vulgui enviar, al Mestre per tal de que les pugui fusionar. Això, però, pot ser molt costós. Aprofitant l'arquitectura paral·lela de la que es disposa val la pena pensar un algoritme paral·lel que permeti fer la fusió.

En la Figura 4.4 es pot veure un exemple del funcionament de l'algoritme amb set nodes: un Mestre, sis Esclaus. L'algoritme proposat és un procés iteratiu. Per entendre'l cal imaginar tots els nodes del clúster (Mestre i Esclaus) en una fila. Cada posició es representa amb un número del 0 a S, on S és el nombre d'Esclaus. El Mestre està en la posició 0. A cada iteració els nodes que estan en una posició imparella envien les seves CS, o subconjunt de d' elements de cada CS, al node parell que tenen immediatament a la seva esquerra. Aquests nodes que han enviat les seves CS deixen de comptar en l'algoritme. Quan un node rep una col·lecció de CS les fusiona amb les seves. En la següent iteració es repeteix el procés, però cal tenir en compte que les posicions han canviat perquè alguns nodes han deixat de comptar. Arriba un moment que el Mestre, el node que està més a l'esquerra, té unes CS resultants de totes les fusions que cal que la enviï a tots els Esclaus.

En el Capítol 5 s'analitzarà aquesta modificació de la paral·lelització.

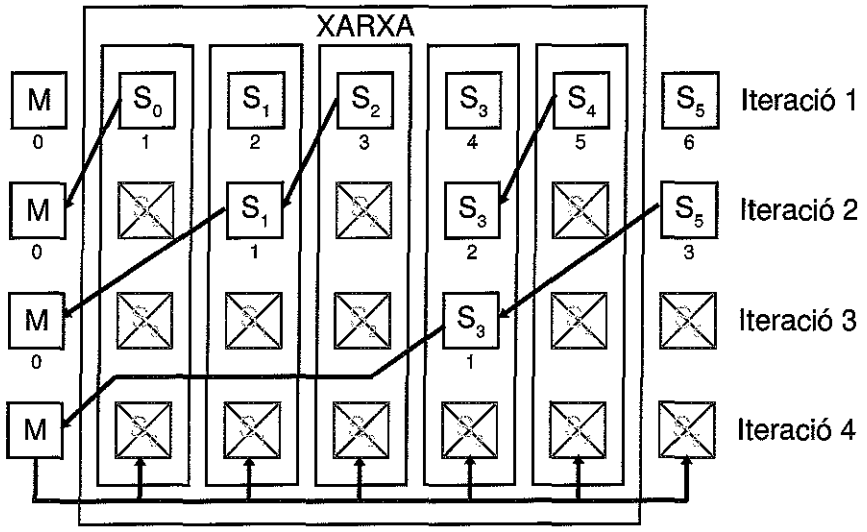

Figura 4.4: Algoritme de fusió de CS. Es mostra quins són els nodes que envien les seves CS, o un subconjunt de d' elements. I quins nodes les reben. Cada cop que un node les rep les fusiona amb les seves. Al final el Mestre(M) té unes CS que són la fusió de les CS de tots els nodes. Cal doncs que les enviï a tots els Esclaus(S_i).

4.7 Conjunt de registres distribuït

A continuació es presenta una modificació de la paral·lelització que a diferència de les altres no té com a objectiu accelerar el procés de Record Linkage paral·lel tant com es pugui, sinó que té com a objectiu poder treballar amb una quantitat de registres més grans.

La Secció 4.2 començava així: "Es suposarà que el conjunt de registres està emmagatzemat en un sol node, el Mestre". Aquesta afirmació, però, té una limitació molt important. La veritat és que s'està suposant que el conjunt de registres cap en un sol node. És a dir, que la memòria d'aquest únic node pot contenir tots els registres. Què passaria, però, si aquesta afirmació no fos certa? En el cas seqüencial caldria emprar tècniques d'out-of-core, però com ja s'ha comentat a l'inici d'aquest document, en el Capítol 1, això és extremadament lent i es descarta d'entrada. En la versió paral·lela, on cada node té la seva pròpia memòria independent, hi ha maneres per poder seguir efectuant el procés suposant que les dades segueixen estan in-core. Una manera de poder contenir aquest conjunt de registres que no hi cap en la memòria d'un sol node, és dividint-lo i repartint-lo entre els diferents nodes.

Aquesta aproximació, però té un seguit d'implicacions en el procés global de Record Linkage. A continuació s'explicaran aquestes implicacions.

El primer problema que cal solucionar és: com es creen els Diccionaris? S'ha dit que per crear els Diccionaris cal tenir tots els registres junts, però ara s'està suposant no és possible. Això presenta un problema. En realitat, per crear els Diccionaris no cal tenir tots els registres junts, el que cal es que un node sigui conscient de tots els registres per poder comptar quants valors diferents hi ha de cada atribut. Així doncs, el Mestre, serà qui carregi els registres en memòria des del fitxer de registres. A mesura que els carregui anirà creant els Diccionaris. Quan hagi carregat N/S registres, on N és el nombre total de registres i S el nombre total d'Esclaus, i hagi omplert els Diccionaris amb la informació d'aquest conjunt de registres, els enviarà a algun Esclau que encara no en tingui. D'aquesta manera al final del procés cada

Esclau tindrà un conjunt de N/S registres i el Mestre tindrà els Diccionaris degudament creats. Això es pot entendre com que tots els registres passen pel Mestre abans d'arribar a l'Esclau que finalment els emmagatzemarà. Com que tots els registres passen pel Mestre aquest pot recollir la informació necessària per crear els Diccionaris.

El següent pas, com fins ara, consisteix en enviar els Diccionaris als Esclaus ja que aquests els empraran per crear les Comparison Stores i els necessitaran per la comparació de registres. Els Esclaus crearan les seves pròpies CS de la mateixa manera que fins ara.

Seguint el procés que es pot veure en la Figura 4.2 el següent pas consisteix en fer l'ordenació per la clau d'ordenació. Aquest segurament és el punt on més afecta el fet de tenir els registres distribuïts. Fins el moment era el Mestre que tenia tots els registres i per tant ordenar-los era quelcom trivial. Ara, però, els registres estan distribuïts, per tant ordenar-los no resulta senzill. En la literatura es poden trobar diferents formes de dur a terme una ordenació amb un conjunt de dades que està distribuït d'una manera eficient [15] [16]. Tot i l'existència d'aquestes tècniques es durà a terme una implementació més rudimentària del procés d'ordenació ja que s'està treballant sobre un software ja existent, DAURUM, i per tant caldria estudiar bé quines implicacions tindria, a nivell de complexitat, adaptar una de les ordenacions proposades en la literatura.

La idea bàsica de l'algoritme d'ordenació que es proposa es basa en el fet que enviar tots els registres al Mestre perquè els ordeni no es possible, ja que no hi caben en memòria. Així doncs cal enviar blocs de registres al Mestre perquè aquest els ordeni i un cop ordenats els enviï a un Esclau. Aquest bloc de registres ordenat resultarà ser la Sfinestra sobre la qual l'Esclau haurà de fer la comparació de registres. A continuació s'analitza amb més profunditat l'algoritme d'ordenació distribuïda.

Cada Esclau té un conjunt de N/S registres. El primer que cal fer és que cada Esclau ordeni el seu conjunt de registres per la clau d'ordenació que toqui. Un cop que cada Esclau hagi ordenat el seu conjunt de registres, enviarà un conjunt de B registres al Mestre. Així doncs el Mestre rebrà S conjunts de B registres i per tant cal que si M és el número màxim de registres que pot contenir un únic node, B sigui com a molt M/S. Cal recordar que cada conjunt de registres està ordenat per la clau d'ordenació. Per tant, el Mestre haurà de comparar el primer registre de cadascun dels conjunts i quedar-se amb el que sigui més petit[1]. Amb els registres que es va quedant anirà construint una Sfinestra. Quan el Mestre s'hagi quedat amb B registres d'un mateix conjunt caldrà demanar a l'Esclau al que pertenyia el conjunt, que si encara té registres per enviar, enviï un nou conjunt de registres. Quan el Mestre hagi pogut construir una Sfinestra sencera de N/S registres l'enviarà a un Esclau. Aleshores el Mestre esborrarà aquesta Sfinestra per alliberar memòria. Val a dir, però, que no l'esborrarà completament, es guardarà els $w-1$ últims registres per construir la propera Sfinestra, que serà de mida $N/S+(w-1)$. Com ja s'ha explicat en la Secció 4.1 cada Sfinestra ha de tenir una superposició de $w-1$ registres amb l'anterior. D'aquesta manera, al finalitzar el procés d'ordenació cada Esclau tindrà la seva Sfinestra sobre la que podrà començar a comparar els registres fent Sliding Window.

Un cop cada Esclau té la seva Sfinestra la resta del procés segueix exactament igual. Es realitzaran t iteracions per augmentar el recall del procés de Record Linkage. A cada iteració es produirà la ordenació distribuïda, cada Esclau seguirà tenint un conjunt de N/S registres, corresponent a la Sfinestra amb la que ha treballat en la darrera iteració. Notis que per fer la següent ordenació distribuïda s'està eliminant de cada Sfinestra la superposició de $w-1$ registres.

La Figura 4.5 mostra un esquema del procés descrit aquí.

[1]Cal entendre que el més petit correspon al que segons la clau d'ordenació va primer.

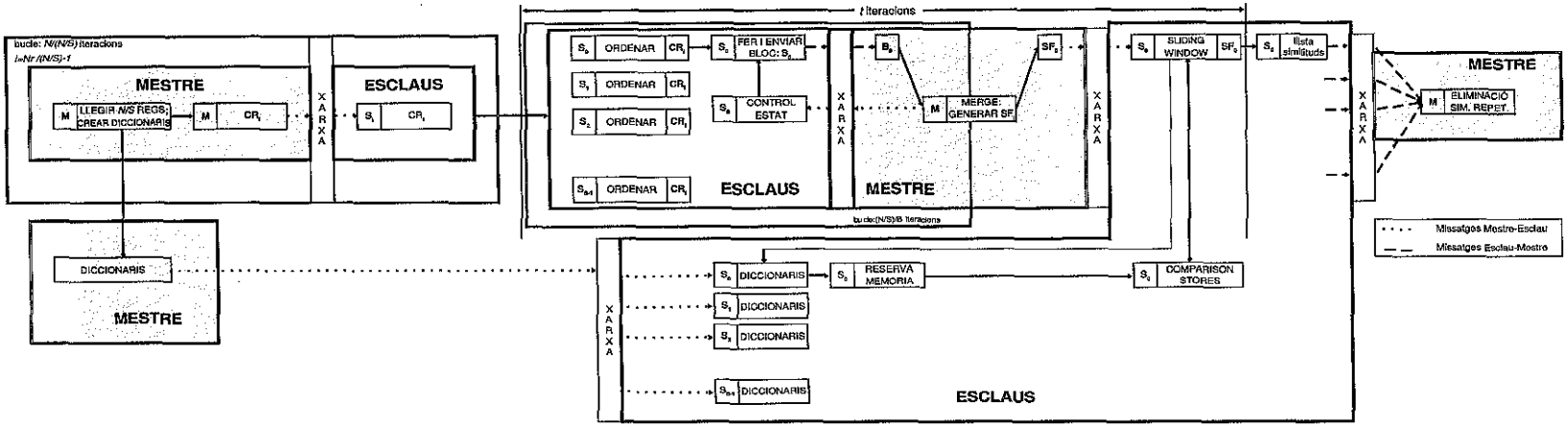

Figura 4.5: Esquema del procés de Record Linkage paral·lel amb un Mestre(M) i S Esclaus(S_i) Per simplificar la imatge es mostra només el comportament de l'Esclau 0. El Mestre càrrega les dades, quan n'ha carregat un conjunt de registres (CR) de N/S registres, on N és el nombre de registres i S el d'Esclaus, envia el CR a S_i on $i = \frac{N_r}{\frac{N}{S}} - 1$, i N_r és el nombre de registres llegits fins el moment $(\frac{N}{S}, \frac{2N}{S}, \cdots, \frac{SN}{S})$. Un cop enviats tots els conjunts de registres el Mestre ha creat els Diccionaris i els envia als Esclaus. Cada Esclau ordena el seu conjunt de registres per la clau d'ordenació. Un cop cada Esclau ha ordenat el seu conjunt de registres, envia blocs de B registres al Mestre perquè aquests els pugui ordenar de manera global. En aquests procés el Mestre va demanant tants blocs B als Esclaus com li faci falta per crear les Sfinestres que anirà enviant als Esclaus. Un cop els Esclaus reben les Sfinestres comencen a fer la comparació dels registres emprant el mètode de Sliding Window. Per fer les comparacions cada Esclau fa ús de la seva col·lecció de Comparison Stores i de Diccionaris. Quan l'Esclau acaba de buscar similituds sobre una Sfinestra avisa al Mestre perquè aquest li pugui enviar una Sfinestra de la següent iteració. Tot aquest procés, d'ordenar, fer Sfinestres, enviar-les i comparar registres, es repeteix t vegades amb diferents claus d'ordenació. Un cop s'han fet les t iteracions cada Esclau envia la llista de similituds que ha trobat al Mestre, que un cop les ha rebudes totes elimina aquelles que estan duplicades.

Es pot observar que aquesta aproximació del procés de Record Linkage paral·lel es força complexa i a nivell computacional té un cost important. És, per tant, d'esperar que resulti força més lenta que la versió paral·lela base. Ara bé, si M és el nombre màxim de registres que pot emmagatzemar un únic node, aquesta versió de la paral·lelització és capaç de treballar amb fins a $S \cdot M$ registres, on S és el nombre d'Esclaus. D'alguna manera es pot parlar de dues aproximacions al problema de la paral·lelització del Record Linkage. Una que està completament orientada a realitzar el procés el més ràpid possible, i per tant accelerar la versió seqüencial tant com es pugui, i l'altra que consisteix en treballar amb grans quantitats de dades. Aquesta segona aproximació tot i que s'espera que sigui més lenta que la paral·lelització original, permet treballar amb un conjunt de dades que d'altra manera no es pot.

En el Capítol 5 s'analitzarà el funcionament d'aquesta versió del Record Linkage paral·lel.

Capítol 5

Experiments

És en aquest capítol on s'explicaran els experiments que s'han fet per avaluar la paral·lelització base que s'ha descrit en el Capítol 4 i les modificacions que s'han fet sobre aquesta primera proposta.

5.1 Introducció

Per avaluar la paral·lelització proposada s'executaran un conjunt d'experiments que mostren la interacció entre el nombre de registres amb el nombre de nodes que executaran el procés de Record Linkage. Per comparar els resultats de la paral·lelització s'emprarà la versió seqüencial de DAURUM, que és un software de Record Linkage que aplica Sliding Window i memoització.

El nombre de registres que s'emprarà per fer els experiments variarà entre 1 milió (1M) i 8 millions (8M), cadascun de 128 bytes. El registres emprats en els experiments s'han generat emprant el generador sintètic de registres inclòs en el joc d'eines FEBRL [9], amb un 30% de duplicats i un màxim de 10 duplicats per registre. Tot i això, per fer l'avaluació el màxim de realista possible, la freqüència dels noms i els cognoms s'han obtingut de l'Idescat. Cada registre conté vuit atributs dels quals quatre són strings: el Nom, el Cognom 1, el Cognom 2 i l'Adreça.

Per realitzar els experiments s'ha usat un clúster de tipus Beowulf [6] amb les característiques descrites en la Taula 5.1.

Clúster Beowulf	
Nombre de nodes:	16
Processador de cada nodes:	Intel Core 2 Duo 6600 @2.4Ghz
Memòria de cada node:	2GB
Mida de cache L2 de cada node:	4096KB
Ampla de banda de la xarxa:	1Gbps

Taula 5.1: Descripció de les característiques del clúster.

5.2 Anàlisi de la paral·lelització base

En aquesta primera secció es veurà com funciona la paral·lelització base. Es començarà fent una anàlisi del temps d'execució del procés de Record Linkage i a continuació s'analitzarà l'eficàcia de la paral·lelització.

5.2.1 Anàlisi del temps

Amb aquests experiments es vol analitzar el temps d'execució del RL. El temps es dissecionarà de la següent manera: 1) el temps en que es comparen els registres, 2) el temps d'eliminació de similituds repetides + el temps de preprocés, que és la fase del procés del RL en la que es creen els Diccionaris i 3) el temps d'**overhead**, que és el preu que es paga per la paral·lelització, és a dir el temps en que s'estan enviant dades a través de la xarxa.

La Figura 5.1 mostra la dissecció per la versió paral·lela amb 2, 4, 8 i 16 nodes. El número de registres als que s'està aplicant el procés de RL està fixat a 8 milions de registres.

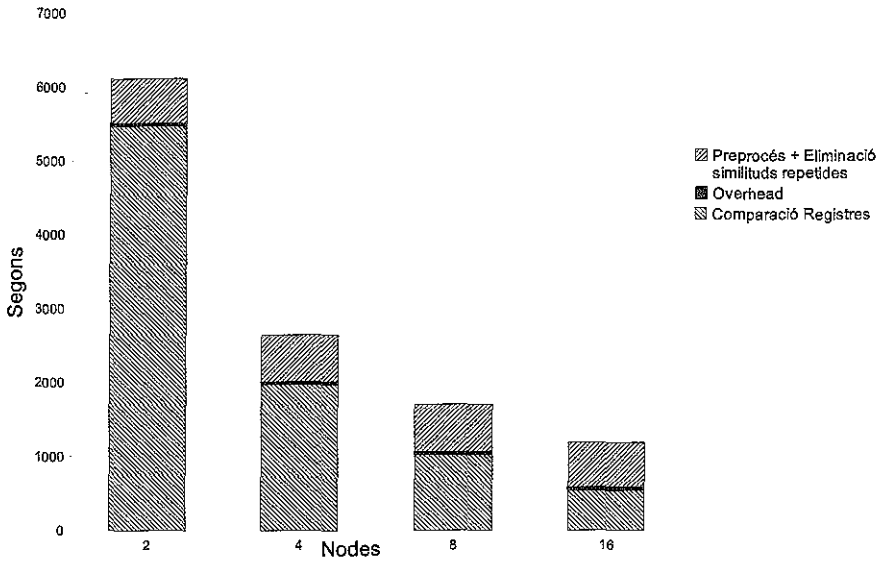

Figura 5.1: Dissecció del temps d'execució del RL per a 8 milions de registres.

S'observa que el temps que RL dedica a la comparació de registres decreix ràpidament a mesura que el nombre de nodes creix ja que aquesta és la fase que s'ha paral·lelitzat. Això vol dir que, arribat a un cert punt el paral·lelisme deixarà de ser eficaç ja que el temps que es triga en comparar registres serà menor que el temps que es triga en d'altres fases. Aquesta afirmació es basa en la llei d'Amdahl [1], que diu que "la millora obtinguda en el rendiment d'un sistema degut a l'alteració d'algun del seus components està limitada per la fracció de temps que utilitza aquest component". Això vol dir que arribat un cert punt en el paral·lelisme resultarà més útil buscar optimitzacions que permetin accelerar el preprocés i l'eliminació de similituds repetides. És interessant observar, però, que gràcies a la paral·lelització s'ha reduït el temps de 100 minuts, més d'hora i mitja, a poc més de 15 minuts.

És també molt interessant observar la influencia de l'overhead sobre el temps d'execució

del procés de Record Linkage paral·lel. Les proves mostren que el temps d'overhead és quelcom negligible en el temps total d'execució. En la Figura 5.2 es mostra el temps d'overhead per a la versió paral·lela de 2, 4, 8 i 16 nodes per a fitxers de registres amb 1, 2, 4 i 8 milions de registres.

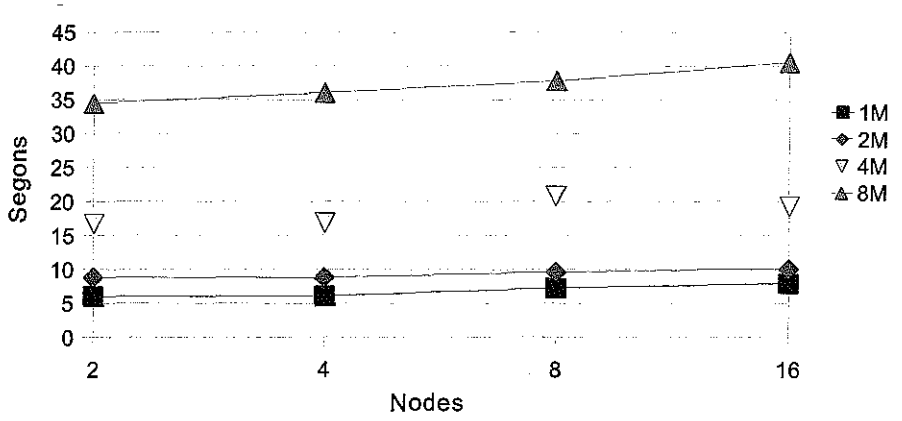

Figura 5.2: Temps d'overhead per a fitxers de registres de 1, 2, 4 i 8 milions de registres amb una paral·lelització amb 2, 4, 8 i 16 nodes.

Es pot veure en la Figura 5.2 que la l'overhead segueix una progressió lineal amb un pendent molt baix. Això vol dir que la tècnica de paral·lelització que s'ha proposat és escalable en el nombre de nodes, almenys amb 16 nodes.

5.2.2 Anàlisi de l'speed-up

L'speed-up, en la computació paral·lela, és el nombre de vegades que algoritme paral·lel és més ràpid que l'algoritme seqüencial. Es defineix a partir de la següent formula $SU_C = T_{seq}/T_C$ on C és el nombre de nodes, T_{seq} és el temps que triga en executar-se l'algoritme seqüencial i T_C és el temps que triga en executar-se l'algoritme paral·lel amb C nodes. Es parla de speed-up lineal o ideal quan $SU_C = C$.

Les execucions que tenen lloc en l'experiment es faran fent servir de 2 a 16 nodes amb uns fitxers de registres d'1 a 8 milions de registres.

Anàlisi de l'speed-up en la Comparació de Registres

Primer s'analitzarà l'speed-up obtingut en la fase en la que es comparen els registres. Notis que no s'està mostrant l'speed-up que s'obté en tot el procés de RL sinó només el de la part que s'ha paral·lelitzat, l'Sliding Window. En la Figura 5.3 es poden observar els resultats obtinguts.

És interessant observar que l'speed-up varia en funció del nombre de registres. En general, com més gran és el número de registres, més gran és l'speed-up. Això és degut a que en l'execució seqüencial com més registres intervenen en el procés més petites són les CS i per tant s'han de fer més comparacions reals. En canvi, en el cas paral·lel la mida de les CS no depenen de la mida del conjunt de registres ja que els Esclaus no emmagatzemen tot el

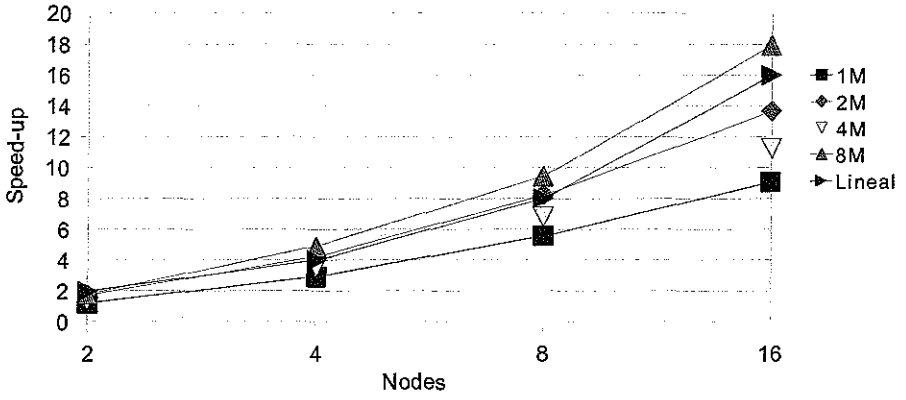

Figura 5.3: Speed-up obtingut en la fase de comparació de registres.

conjunt de registres. Això explica la gran diferència entre l'speed-up, amb 16 nodes, obtingut amb el conjunt de registres d'1 milió i l'obtingut amb el de 8 milions. Per entendre la petita diferencia que es produeix entre l'speed-up, amb 16 nodes, obtingut amb el conjunt de registres de 2 milions i l'obtingut amb el de 4 milions, que va en contra del que s'ha definit com a comportament general de l'speed-up, cal tenir en compte el que s'ha explicat en la Secció 4.6. S'ha explicat que en la versió paral·lela es produeixen en total més comparacions reals que en la versió seqüencial. Depenent, doncs, del contingut dels registres, la mateixa comparació es durà a terme en més o menys nodes i l'speed-up es veurà afectat.

En general s'observa que en mitja s'obté un speed-up força bo. L'eficàcia d'una paral·lelització es defineix com $E_C = SU_C/C$. En la Taula 5.2 es pot veure que l'eficàcia de la paral·lelització en el millor cas és del 122%, una eficàcia més alta que la lineal. I que la mitja de les eficàcies és de gairebé un 90%, el que és una dada més que positiva.

	2 nodes	4 nodes	8 nodes	16 nodes	Màxima eficàcia/ número registres
1 milió	0,6	0,73	0,7	0,57	0,73
2 milions	0,87	1,06	1,03	0,85	1,06
4 milions	0,69	0,88	0,85	0,7	0,88
8 milions	0,88	1,22	1,18	1,12	1,22
Màxima eficàcia	1,22				
Eficàcia mitja	0.87				

Taula 5.2: Eficàcia de la paral·lelitzacions per diferents quantitats de registres i diferents nodes.

Anàlisi de l'speed-up en el procés de RL

Un cop vist l'speed-up que s'obté en la comparació de registres cal veure com influeix en l'speed-up total del procés. En la Figura 5.4 es pot veure l'speed-up que s'obté en el procés de Record Linkage.

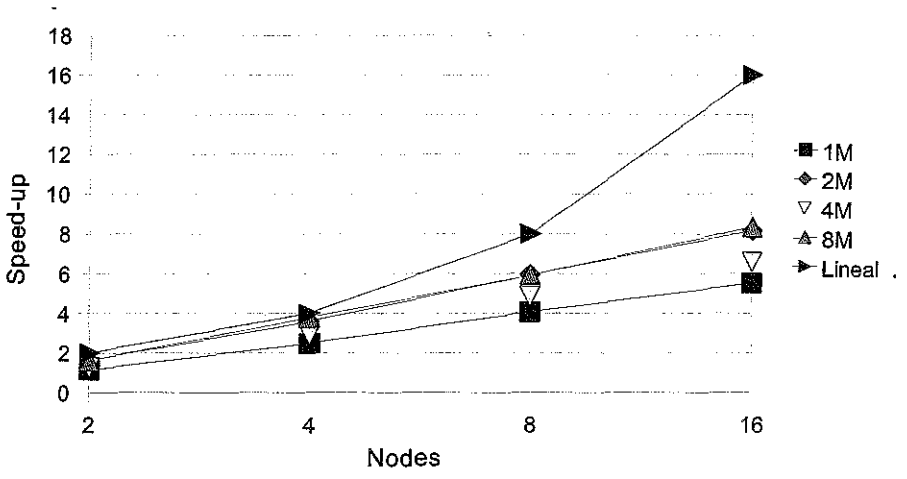

Figura 5.4: Speed-up obtingut en el procés de Record Linkage.

El resultat que s'observa era d'esperar. L'speed-up s'allunya molt més del lineal del que s'ha vist en la Figura 5.3. Això és degut a que només s'està paral·lelitzant una part del procés. De fet, s'observa que com més nodes intervenen en el procés més s'allunyen de l'speed-up lineal. L'explicació a aquest comportament és senzilla i és conseqüència de la llei d'Amdahl, com més nodes intervenen, la part no paral·lelitzada té un pes més important en total de l'execució com es mostra en la Figura 5.1 i per tant la reducció del temps de comparació de registres no té un gran impacte en el temps total.

Tot i això, s'arriba a efectuar el procés de Record Linkage fins a 8 cops més ràpid, quan es treballa amb gran quantitat de registres, que en la versió seqüencial.

5.3 Comparació Mestre-Treballador i Mestre-Gandul

Cal començar recordant que la versió Mestre-Treballador és aquella versió en la que el node Mestre efectua també el rol d'Esclau. Això vol dir que, donat un número de nodes la versió Mestre-Treballador té un Esclau més que en la versió Mestre-Gandul, en la que el node Mestre només fa de Mestre. Per tant, en la versió Mestre-Treballador hi ha una Sfinestra més que en la versió Mestre-Gandul i per tant cada Sfinestra té un número menor de registres.

La Figura 5.5 mostra el temps que ha trigat en efectuar-se el procés de RL per a diferents mides del conjunt de registres. El nombre de processadors està fixat a 16 nodes. Per tant la versió Mestre-Treballador tindrà 1 Mestre i 16 Esclaus, i la Mestre-Gandul 1 Mestre i 15 Esclaus.

Es pot observar que fins a 4 milions de registres la versió Mestre-Treballador és sensiblement més ràpida que la versió Mestre-Gandul. Ara bé, a partir dels 8 millions de registres,

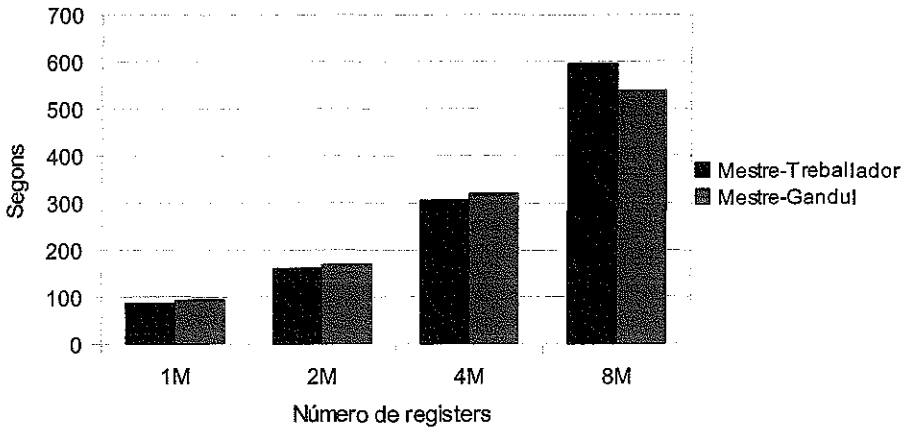

Figura 5.5: Temps d'execució del procés de Record Linkage per a diferents quantitats de registres en la versió Mestre-Treballador i Mestre-Gandul.

la versió Mestre-Gandul funciona millor. Per entendre el perquè d'aquest fet cal entrar, en detall, en què vol dir exactament fer servir Mestre-Gandul o Mestre-Treballador.

En la versió Mestre-Gandul, mentre els Esclaus estan comparant els registres el Mestre ja pot estar efectuant la següent ordenació de la iteració i dividint el conjunt de registres en Sfinestres. D'aquesta manera, quan un Esclau ha acabat de comparar els registres d'una Sfinestra i avisi al Mestre que ja ha acabat, el Mestre li pot enviar directament una nova Sfinestra perquè ja la té preparada.

En la versió Mestre-Treballador, mentre els Esclaus estan comparant els registres de les seves Sfinestres el Mestre ha d'estar comparant els registres de la seva pròpia Sfinestra. Això vol dir que, quan un Esclau ha acabat de comparar els registres d'una Sfinestra i avisi al Mestre que ja ha acabat, haurà d'esperar a que el Mestre acabi de fer les comparacions, ordeni els registres per la següent clau d'ordenació, divideixi els registres en Sfinestres i, finalment, li enviï.

Resulta curiós, un cop vist això, que en alguns casos i depenent de la mida del conjunt de registres sigui millor la versió Mestre-Treballador i en d'altres la Mestre-Gandul. Per acabar-ho d'entendre val la pena pensar en termes de Comparison Store. En la Figura 5.6 es pot veure la memòria total que han reservat els Esclaus per a les Comparison Stores. Notis que la memòria total reservada pels diferents Esclaus en la versió Mestre-Treballador i Mestre-Gandul és la mateixa i per tant només apareix un cop. Per altra banda també es pot veure la memòria total reservada pel Mestre, només en el cas de Mestre-Treballador ja que en la versió Mestre-Gandul, com que no ha de fer comparacions entre registres no necessita les CS. Es pot observar que la memòria dels Esclaus és la mateixa independentment del número total de registres que hi ha. Contrariament, la memòria reservada pel Mestre decreix a mesura que el número de registres és major. Per entendre això cal recordar dos fets: a) el Mestre és l'únic que que conté tots els registres i b) les CS reserven tanta memòria com poden. Això vol dir que com més gran sigui el conjunt de registres que ha d'emmagatzemar menys memòria disponible tindrà per les Comparison Stores, i per tant seran més petites. Això tindrà com a conseqüència que en la fase de comparació de registres haurà de fer més comparacions reals entre atributs i per tant serà més lent en efectuar-la.

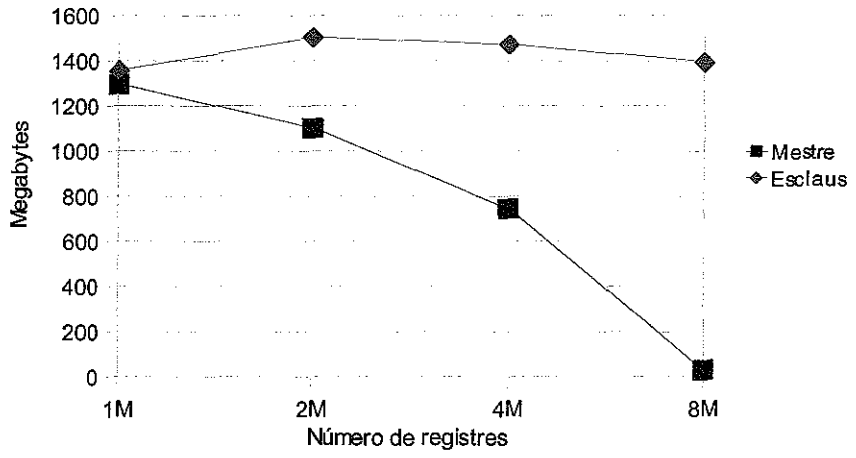

Figura 5.6: Memòria reservada per les Comparison Stores per a diferents mides del conjunt de registres pels Esclaus i pel Mestre.

Es conclou, doncs, que fins a un determinat número de registres és una bona opció fer servir la versió Mestre-Treballador perquè com que les Sfinestres sobre les que s'ha de treballar són més petites, el procés en general es veu accelerat. Passat aquest cert punt fer servir la versió Mestre-Gandul és una millor idea ja que a mesura que el número de registres creix, la mida de les CS del Mestre decreixen, i decreixen tant que arriba un punt que triga tant en comparar els registres de la seva Sfinestra, que quan acaba de fer-ho els Esclaus fa estona que s'esperen, sense fer res, a que els enviïn una nova Sfinestra sobre la que comparar registres, fet que provoca que el procés de RL sigui més lent.

5.4 Anàlisi speed-up de la paral·lelització a nivell de node

L'objectiu de paral·lelitzar a nivell de node és clarament augmentar l'speed-up obtingut quan només es fa la paral·lelització distribuïda. Així doncs, els següents experiments tenen com a objectiu analitzar l'efectivitat que s'obté pel fet de realitzar una paral·lelització SMP a cadascun dels nodes. Per mesurar-ho es compararan els speed-ups obtinguts en la versió on s'executa el procés de RL amb únicament 1 thread (paral·lelització base), i els speed-ups obtingut quan s'executa el procés amb 2 threads, que són el número de cores que tenen els ordinadors que configuren els clúster sobre el que s'estan fent els experiments.

Els experiments s'han realitzat sobre un conjunt de registres de 8 milions amb 2, 4, 8 i 16 nodes amb 1 i 2 threads.

En les Figures 5.7 i 5.8 es mostren els speed-ups obtinguts en la fase on es comparen els registres i en tot el procés de Record Linkage sencer respectivament.

Els resultats mostren que la paral·lelització a nivell de node resulta molt eficaç. Es pot observar que l'speed-up obtingut en el cas d'utilitzar 2 threads gairebé doblen l'speed-up obtingut utilitzant únicament 1 thread. És a dir, estan molt a prop de l'speed-up lineal si es pren com a valors base els obtinguts per la paral·lelització amb 1 thread. Cal destacar que s'està arribant, doncs, a executat la fase de comparació de registres fins a trenta-dues vegades més ràpid i fins a gairebé setze vegades més ràpid tot el procés de RL.

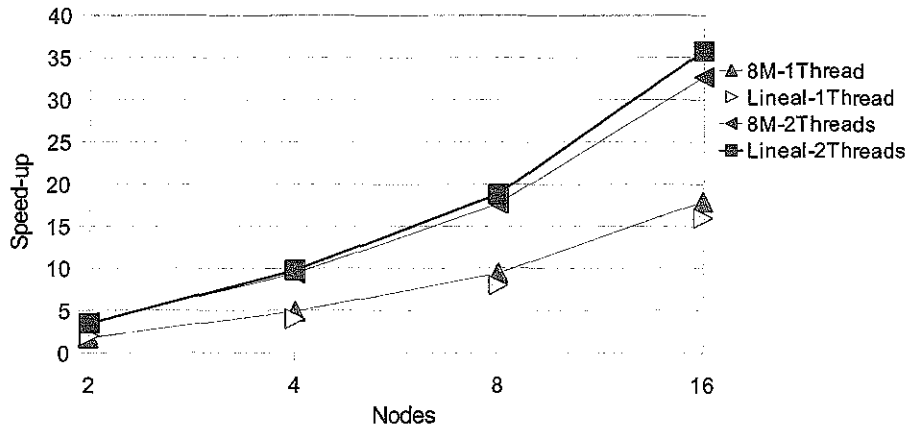

Figura 5.7: Speed-up obtingut en la comparació de registres amb 1 i 2 threads.

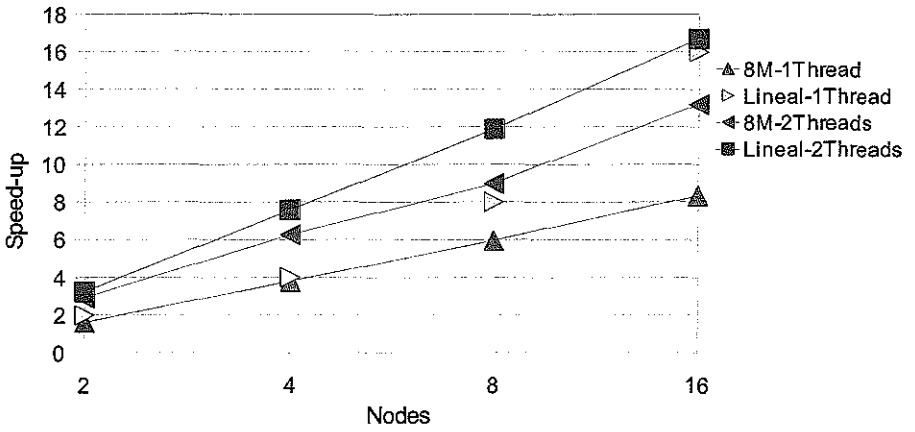

Figura 5.8: Speed-up obtingut en el procés de Record Linkage amb 1 i 2 threads.

Aquest experiment recolza el fet d'haver escollit una paral·lelització distribuïda ja que els ordinadors de sobretaula, emprats en la configuració del clúster, ja incorporen, quelcom que avui en dia resulta molt normal, i a mesura que passi el temps encara ho resultarà més, diferents processadors integrats. Això el que acaba permetent és realitzar una paral·lelització eficaç a dos nivells diferents i a baix cost.

5.5 Anàlisi de la compartició del contingut de les CS

La versió de la paral·lelització que comparteix el contingut de la CS té com a únic objectiu accelerar el procés de RL mitjançant la disminució global del número de comparacions.

Els experiments es faran amb 16 nodes ja que és el cas en el que una mateixa comparació es pot efectuar més cops, una en cada node, i per tant el cas en que es produeixin en global més comparacions.

Per comparar ambdues versiones es mostrarà l'speed-up obtingut pel fet de compartir el contingut de les CS. Aquest speed-up es calcula de la següent manera:
$SU = T_{sense\ compartir}/T_{compartint}$, on $T_{sense\ compartir}$ és el temps que triga el procés de Record Linkage en executar-se amb la paral·lelització base i $T_{compartint}$ és el temps que triga en executar-se el procés de Record Linkage amb la paral·lelització en la que es comparteix el contingut de les Comparison Stores.

En la Figura 5.9 es pot observar l'speed-up obtingut mitjançant la compartició dels resultats de les comparacions realitzades entre els 5.000, 10.000, 20.0000 i 30.000 strings més freqüents de cada CS, en la primera iteració del procés, per a 1, 2, 4 i 8 milions de registres en la fase de comparació de registres, respecte a la versió en que no es comparteix el contingut de les CS. Per calcular aquest speed-up no s'ha tingut en compte l'overhead que la compartició suposa, és a dir, tenir un control sobre les comparacions que s'han realitzat, enviar-les a través de la xarxa i fusionar les CS.

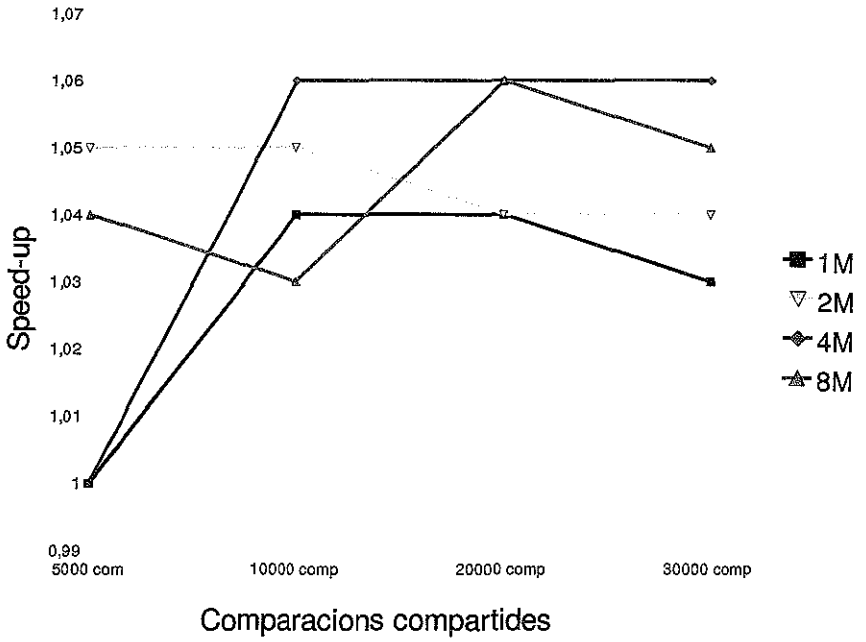

Figura 5.9: Speed-up respecte la paral·lelització que no comparteix el contingut de les CS en la fase de comparació de registres obtingut a partir de la compartició dels 5.000, 10.000, 20.0000 i 30.000 strings més freqüents de cada CS, per a 1, 2, 4 i 8 milions de registres. No es té en compte l'overhead.

S'observa que en els millors dels cassos l'speed-up és d'un 6%. Malgrat que no és un gran speed-up, sí que resulta interessant. Cal observar, també, que el millor speed-up s'aconsegueix, en general, amb el conjunt de 4 milions de registres. Això recolza l'explicació que s'ha donat en la Secció 5.2.2 en la que s'ha explicat que, en general, com més gran la quantitat de registres millor l'speed-up, tot i això s'observava una anomalia en el cas del conjunt de registres de 4 milions en que l'speed-up era lleugerament pitjor que en el cas de 2 milions de registres. S'ha justificat recordant el que s'havia explicat en la Secció 4.6; que el fet que en

la paral·lelització es produeixin en total més comparacions que en la versió seqüencial podia afectar més o menys a l'speed-up final en funció del conjunt de registres. El fet que el millor speed-up s'aconsegueixi amb el conjunt de registres de 4 milions recolza, doncs, el fet que aquest conjunt en particular es veu força afectat per les repeticions addicionals que s'efectuen com a causa de la paral·lelització.

Tot i que l'speed-up de la comparació de registres resulta interessant, no es pot concloure res, sense veure com afecta l'overhead en l'speed-up obtingut. Aquest speed-up es mostra en la Figura 5.10.

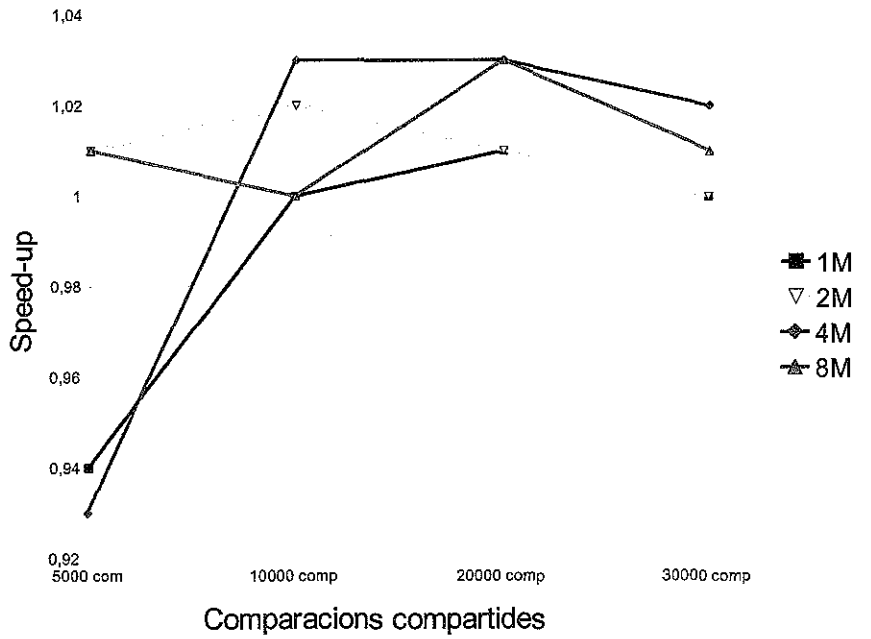

Figura 5.10: Speed-up respecte la paral·lelització que no comparteix el contingut de les CS en la fase de comparació de registres obtingut a partir de la compartició dels 5.000, 10.000, 20.0000 i 30.000 strings més freqüents de cada CS, per a 1, 2, 4 i 8 milions de registres. Es té en compte l'overhead.

S'observa que en els millors dels casos l'speed-up és d'un 3%. És a dir que l'speed-up aconseguit es veu completament contrarestat per l'overhead que suposa el fet d'haver de compartir el contingut de les CS, és a dir, dur un control de quines comparacions s'han efectuat, i fusionar les Comparison Stores.

Es pot concloure que la compartició del contingut de les CS entre els diferents Esclaus és quelcom que pot donar resultats interessant però no espectaculars, doncs, s'està arribant a obtenir un speed-up de fins a un 6%. Cal, però, trobar d'altres maneres més eficients que la que s'ha proposat en el projecte per tal de que l'overhead que la compartició de les CS genera no contraresti l'speed-up que s'obté.

5.6 Anàlisi de la paral·lelització amb el conjunt de registres distribuït

Amb aquests experiments es vol analitzar el funcionament de la versió de la paral·lelització en la que el conjunt de registres està distribuït entre tots els Esclaus que estan implicats en el procés de RL paral·lel.

Donada la importància d'aquesta modificació de la paral·lelització base ja que més que una modificació és una aproximació diferent al problema, que té com a objectiu poder treballar amb molts més registres, els experiments seguiran el mateix esquema que els de la paral·lelització base. Primer es farà una anàlisi del temps d'execució del procés de Record Linkage i a continuació mostrarà l'speed-up obtingut arran de la paral·lelització.

Per poder comparar aquesta versió de la paral·lelització amb la paral·lelització base els experiments es faran amb un conjunt de 8 milions de registres que és el màxim número de registres que pot emmagatzemar un sol node dels del clúster que s'està emprant. Malgrat que s'ha vist que aquesta quantitat de registres sí que cap en un sol node, es suposarà que no i es distribuirà.

5.6.1 Anàlisi del temps

Amb aquests experiments es vol analitzar els diferents components de temps que conformen el procés de RL paral·lel amb els registres distribuïts. El temps es disseccionarà de la següent manera: 1) temps de comparació de registres, 2) temps de preprocés + temps d'eliminació de similituds repetides, 3) temps d'overhead i 4) temps de la ordenació, que és el temps que s'ha emprat en ordenar el conjunt de registres distribuïts.

En la Figura 5.11 es pot veure la dissecció del temps per la paral·lelització base i la paral·lelització amb els registres distribuïts (-RD), amb 4, 8 i 16 nodes. Notis que no s'ha fet servir la configuració amb 2 nodes, ja que tal i com s'ha descrit el procés de distribució de registres, tots els registres els hauria d'emmagatzemar l'únic Esclau disponible, però s'està suposant que no hi cap en un únic node.

Arran d'aquests experiments es poden concloure diferents qüestions.

1 És interessant veure que l'overhead que es produeix segueix sent completament negligible i s'espera per tant que segueixi el mateix comportament que el que es pot observar en la Figura 5.2. Per tant, des del punt de vista de l'overhead, és pot dir que la paral·lelització és escalable en el número de nodes, almenys fins a 16 nodes.

2 El temps que es dedica en fer l'ordenació no varia de manera apreciable en funció del nombre de nodes en els que el conjunt de registres està distribuït. Això indica que, des del punt de vista de l'ordenació distribuïda, la paral·lelització també és escalable en el número de nodes, almenys fins a 16 nodes.

També es pot observar que la diferència en el temps de comparació de registres entre les dues versions de la paral·lelització decreix a mesura que el número de nodes creix. Per entendre això cal tornar a pensar en termes de Comparison Stores i tenir molt en compte la Figura 5.6 en la que es mostra que la mida de les CS està estretament lligada amb el número de registres que s'estan emmagatzemant. Així doncs, en la versió base de la paral·lelització el conjunt de registres estarà emmagatzemat en el Mestre i la CS dels Esclaus seran independents de la quantitat de registres. Ara bé, en la versió de la paral·lelització en la que els registres estan distribuïts, cada Esclau contindrà N/S registres. Donada una quantitat fixa

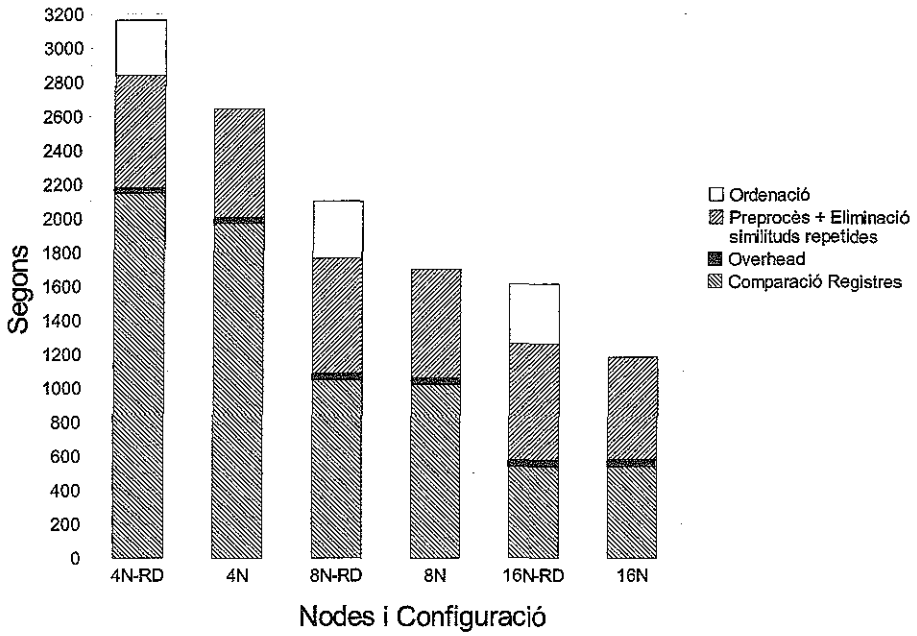

Figura 5.11: Dissecció del temps d'execució del RL per a 8 milions de registres de la paral·lelització base i la paral·lelització amb registres distribuïts.

de registres, com més Esclaus hi hagi, menys registres haurà d'emmagatzemar cadascun i per tant les seves CS seran més grans i el temps que triguen en comparar els registres disminuirà, apropant-se, així, al temps que es triga en la paral·lelització base.

Per últim, és interessant comentar el temps de Preprocés + Eliminació de les similituds repetides. En el cas de 16 nodes (16N) es pot observar clarament que aquest temps és lleugerament major en la versió de registres distribuïts que en la versió base de la paral·lelització. Això és degut al temps de preprocés. Tal i com s'ha vist en la Secció 4.7, el preprocés de la versió amb els registres distribuïts és més costos que el de la paral·lelització base ja que tots els registres han de passar pel Mestre abans d'arribar a l'Esclau que finalment els emmagatzemarà.

Per altra banda, les mateixes conclusions a les que s'havia arribat amb la versió base de la paral·lelització segueixen tenint validesa. El temps que es dedica a la comparació de registres deixa de ser el més rellevant en el conjunt de l'execució i per tant cal buscar d'altres optimitzacions que permetin accelerar el preprocés i l'eliminació de similituds repetides i/o l'ordenació, mitjançant, per exemple i com s'ha comentat en la Secció 4.7, tècniques més avançades d'ordenació paral·lela [15] [16].

5.6.2 Anàlisi de l'speed-up

Els següents experiments tenen com a objectiu veure quin és l'speed-up que s'obté a partir de la paral·lelització que té el conjunt de registres distribuïts, així com comparar aquest speed-up amb els de la versió base de la paral·lelització.

S'executarà el procés de Record Linkage amb un conjunt d'entrada de 8 milions de registres amb diferent número de nodes, de 4 a 16.

En la Figura 5.12 s'observa l'speed-up obtingut en la comparació de registres en les dues

propostes de paral·lelització. S'observa que, a mesura que augmenta el número de nodes que intervenen en el procés, l'speed-up obtingut per ambdues versions de la paral·lelització convergeix. Això va en consonància amb el que s'ha explicat en els anteriors experiments, on s'ha dit "la diferència entre els temps de comparació de registres, entre les dues versions de la paral·lelització, decreix a mesura que el número de nodes creix".

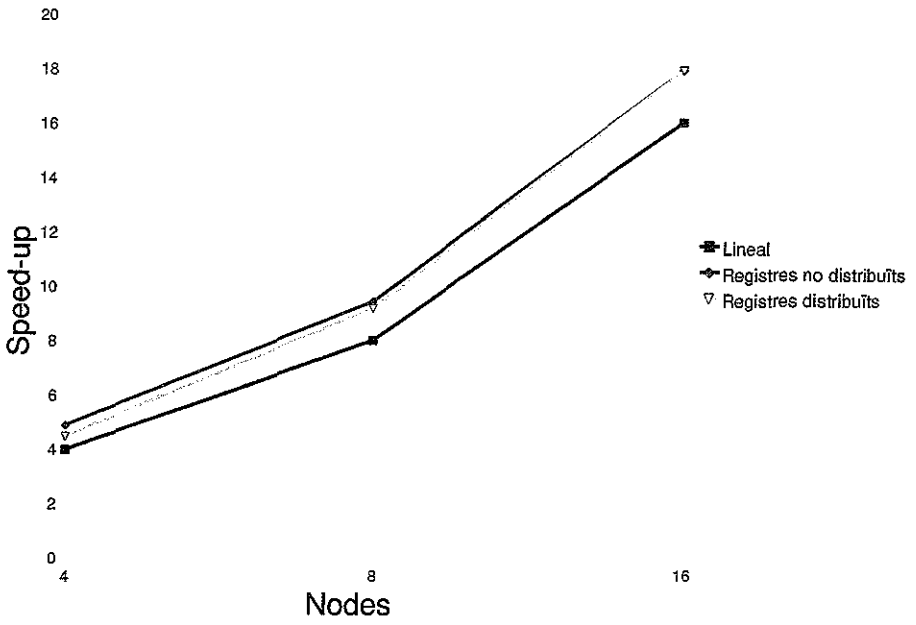

Figura 5.12: Speed-up obtingut en la comparació de registres amb la versió base de la paral·lelització i la versió que té els registres distribuïts.

La Figura 5.13 mostra l'speed-up obtingut en tot el procés de RL mitjançant la paral·lelització base i la paral·lelització amb els registres distribuïts. S'observa que com més nodes intervenen en el procés més s'allunya l'speed-up aconseguit de l'speed-up lineal. Aquest fet s'explica de la mateixa manera que s'explicava en la Secció 5.2.2: només s'està paral·lelitzant una part del procés sencer de RL, per tant com més nodes intervenen més petita és aquesta part i més pes té la part no paral·lelitzada. Per altra banda també es pot veure que a mesura que intervenen més nodes l'speed-up aconseguit s'allunya lleugerament de l'speed-up aconseguit per la paral·lelització base. Això es deu bàsicament al temps que es dedica en l'ordenació i en el preprocés que, tal i com s'ha comentat arran de la Figura 5.11 és més costós en la versió amb els registres distribuïts que en la versió base. Tot i això, aquesta versió és capaç d'executar fins a sis cops més ràpid el procés de Record Linkage seqüencial amb 8 milions de registres emprant 16 nodes i de treballar amb fins a S vegades més de registres que la versió seqüencial i la paral·lelització base.

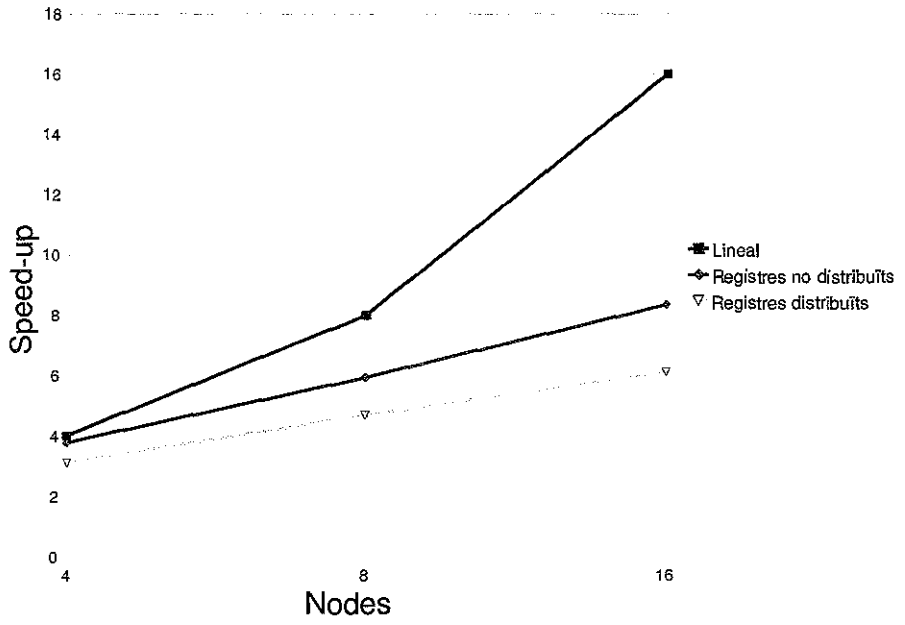

Figura 5.13: Speed-up obtingut en el procés de Record Linkage amb la versió base de la paral·lelització i la versió que té els registres distribuïts.

Capítol 6

Planificació i Costos

Un cop vist tot l'estudi, el disseny, la implementació i els experiments de la paral·lelització proposada, i les diverses modificacions que s'han fet per complementar-la, resulta interessant parlar dels aspectes més relacionats amb la gestió del projecte pel que fa a la planificació i les despeses associades a la seva realització. Aquest capítol malgrat que no té un contingut tècnic es pot dir que és vital en l'àmbit de l'enginyeria ja que sense tenir una planificació sobre la que recolzar-se, ni una viabilitat econòmica del projecte resulta molt difícil dur-lo a terme.

6.1 Planificació del projecte

La planificació d'un projecte com el que s'ha descrit no és quelcom estàtic que es dugui a terme al principi d'aquest i després romangui inalterable. És quelcom viu, que cal anar revisant i modificant a mesura que s'avança en el projecte. Cal tenir en compte que la realitat en l'execució del projecte fins a un determinat moment pot distar, fins a cert punt, del que s'havia planificat i per tant afecta a la planificació de la resta del projecte, que s'ha de veure modificada per corregir dites desviacions.

És pot dir, per tant, que la planificació a l'inici d'un projecte és la previsió de temps de cada tasca. A partir d'aquesta planificació sorgeix un pressupost que marca la viabilitat econòmica del projecte. Aquesta planificació inicial, que resulta d'una previsió en base a l'experiència, a mesura que passa el temps cal que canviï adaptant-se a les desviacions comesses durant el decurs del projecte de manera que acabi resultant una descripció real de les hores que s'han destinat, per poder-ne, així, calcular el cost final.

Tot i que es podria mostrar la planificacio original que hi va haver i, com s'ha anat modificat per dur terme les correccions necessàries per adaptar-se a les desviacions i imprevistos, per no estendre innecessàriament el capítol, s'opta per mostrar el resultat final de la planificació després d'haver finalitzar el projecte.

Diagrama de Gantt

A continuació, en la Figura 6.1 es mostra un diagrama de Gantt en el que es divideix tot el projecte en les diferents tasques, i per cadascuna mostra els terminis en que s'ha dut a terme.

Notis que el fet de que aquest projecte s'hagi desenvolupat en el context d'un grup de recerca ha marcat molt el perfil de la planificació seguin un procés de d'anàlisi, dissenys, implementació i experiments.

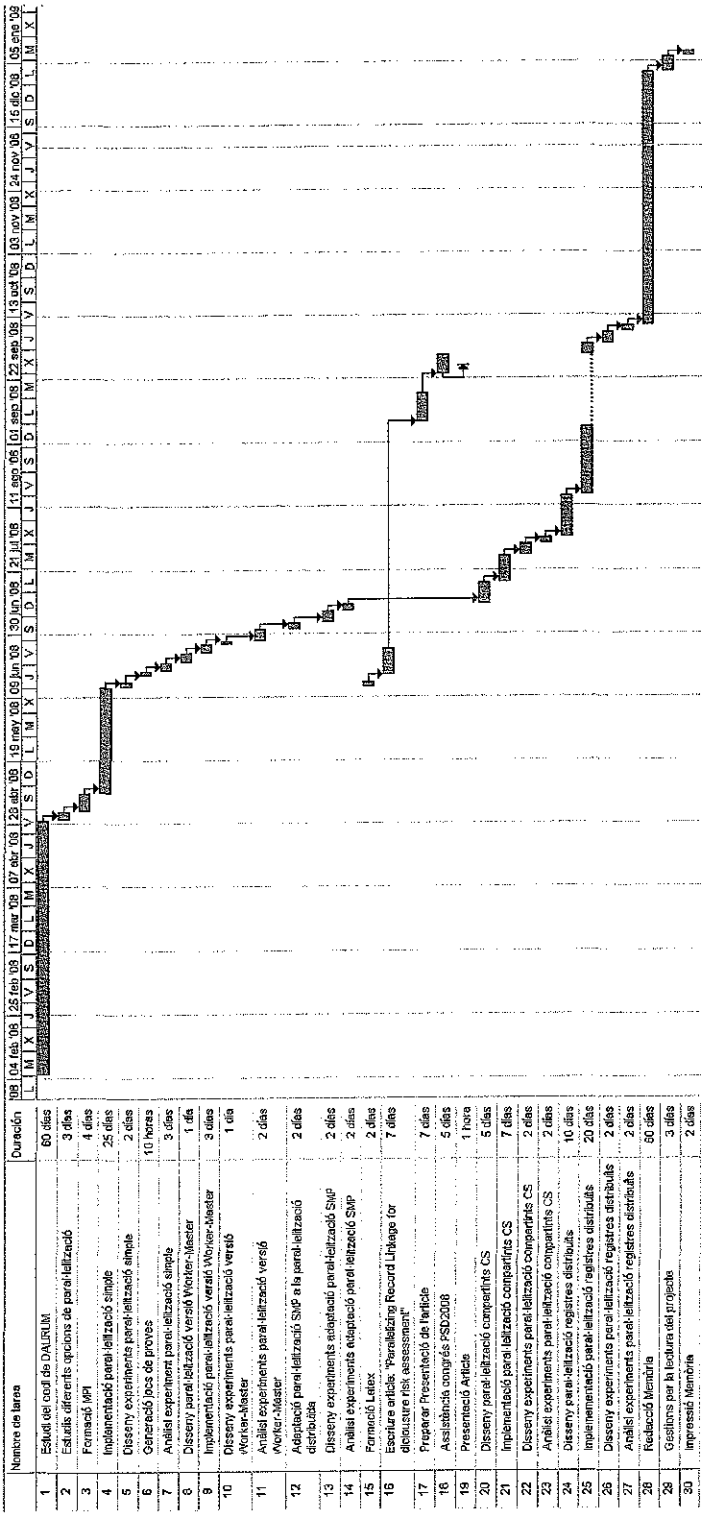

Figura 6.1: Diagrama de Gantt.

6.2 Costos del projecte

De la mateixa manera que en la planificació temporal, és molt important realitzar un pressupost en iniciar el projecte, que permet-hi fer-se a la idea de l'envergadura del projecte des

d'un punt de vista econòmic. Aquests pressupost ha de permetre saber si es disposen de suficient recursos com per poder realitzar el projecte.

És, però, al finalitzar el projecte quan es pot realitzar, en base a la realitat, l'estat de comptes. L'estat de comptes reflectirà el cost real que ha tingut el projecte.

Dins dels costos, es poden distingir dos grans blocs. Per una banda els costos personals, que són els recursos humans que s'encarreguen de dur a terme totes les fases del projecte i per altra banda, els recursos materials, que són les eines necessàries per realitzar les tasques.

En la configuració del cost total del projecte solent intervenir-hi d'altres factors, com poden ser despeses de desplaçament, sistema elèctric, desamortització de les eines, imprevistos... Aquest factors, força més complexes de calcular, no es tindran en compte en el càlcul final del cost del projecte.

6.2.1 Recursos humans del projecte

La partida en la que es destinen més recursos econòmics és en la de personal. Aquest projecte, com resulta obvi al ser un projecte final de carrerar, no té un equip de personal especialitzat en cadascuna de les tasques, sinó que hi ha una única persona, el projectista, que s'encarrega de dur a terme totes elles, a part del Director del projecte que, per descomptat, ajuda en la realització del mateix.

Per mostrar un càlcul de costos destinat als recursos humans més realista es simularà un conjunt de personal que s'encarregaran de dur a terme les tasques explicitades en el diagrama de Gantt.

En el desenvolupament d'un projecte informàtic intervenen diferents rols, el cap de projecte, l'analista, el dissenyador i el programador. Cadascun d'aquests rols es fa càrrec d'una part del projecte segons les seves competències.

En aquest projecte es suposarà que la figura del cap de projecte està duta a terme per l'analista, que a més a més, és l'encarregat de la part d'anàlisi i especificació de requisits. Això permetrà simplificar la comprensibilitat de la secció, així com simplificar l'assignació de recursos. El dissenyador és l'encarregat de fer el disseny del sistema software i el programador és l'encarregat de la codificar-lo. En la Taula 6.1 es mostren el preu per hora fixat per a cada escala laboral.

Nom del recurs	Tarifa
Analista	40 €/hora
Dissenyador	35 €/hora
Programador	20 €/hora

Taula 6.1: Tarifes en €/hora per les diferents escales laborals.

A partir del calendari del projecte, l'assignació de recursos humans a cada tasca i el salari per a cada treballador és possible realitzar el càlcul del cost total del projecte que es pot veure en la Taula 6.2.

Nom de la tasca	**Recurs Personal**	**Hores de feina**	**Cost total**
Estudi del codi de DAURUM	Analista Dissenyador	480 hores	18.000,00 €
Estudi diferents opcions de paral·lelització	Analista Dissenyador	24 hores	900,00 €

Formació MPI	Dissenyador Programador	32 hores	880,00 €
Implementació paral·lelització base	Programador Dissenyador	100 hores	2.750,00 €
Disseny experiments paral·lelització base	Analista Dissenyador	16 hores	600,00 €
Generació jocs de proves	Programador	10 hores	200,00 €
Anàlisi experiment paral·lelització base	Analista	24 hores	960,00 €
Disseny paral·lelització versió Mestre-Treballador	Dissenyador	8 hores	280,00 €
Implementació paral·lelització versió Mestre-Treballador	Programador	24 hores	480,00 €
Disseny experiments paral·lelització versió Mestre-Treballador	Dissenyador Analista	4 hores	150,00 €
Anàlisi experiments paral·lelització versió Mestre-Treballador	Analista	16 hores	640,00 €
Adaptació paral·lelització SMP a la paral·lelització distribuïda	Programador Dissenyador	16 hores	440,00 €
Disseny experiments adaptació paral·lelització SMP	Dissenyador	16 hores	560,00 €
Anàlisi experiments adaptació paral·lelització SMP	Analista	16 hores	640,00 €
Formació LaTeX	Analista	16 hores	640,00 €
Escriure article: Parallelizing Record Linkage for disclousure risk assessment	Analista	56 hores	2.240,00 €
Preparar Presentació de l'article	Analista	56 hores	2.240,00 €
Assistència congrés PSD2008	Analista	40 hores	1.600,00 €
Presentació Article	Analista	1 hora	40,00 €
Disseny paral·lelització compartint CS	Dissenyador	40 hores	1.400,00 €
Implementació paral·lelització compartint CS	Programador	56 hores	1.120,00 €
Disseny experiments paral·lelització compartint CS	Analista Dissenyador	16 hores	600,00 €
Anàlisi experiments paral·lelització compartint CS	Analista	16 hores	640,00 €
Disseny paral·lelització registres distribuïts	Dissenyador	80 hores	2.800,00 €
Implementació paral·lelització registres distribuïts	Programador	160 hores	3.200,00 €
Disseny experiments paral·lelització registres distribuïts	Dissenyador Analista	16 hores	600,00 €
Anàlisi experiments paral·lelització registres distribuïts	Analista	16 hores	640,00 €

Redacció Memòria	Analista Dissenyador	480 hores	18.000,00 €
Gestions per la lectura del projecte		0 hores	0,00 €
Impressió Memòria		0 hores	0,00 €
TOTAL		1835 hores	63.240,00 €

Taula 6.2: Cost total del projecte en funció de l'assignació de cada tasca als diferents treballadors i les hores que se li han dedicat.

S'observa que el desenvolupament del projecte ha necessitat de 1.835 hores de feina, i que el conjunt de les tasques que s'han desenvolupat han tingut un cost de 63.240 €.

6.2.2 Recursos materials

Cal tenir en compte, també, els costos que es deriven de la utilització del maquinari i de software.

Pel que fa el software que s'ha emprat per l'execució del projecte val a dir que s'ha fet servir tot de software emmarcat dins del programari lliure. El sistema operatiu sobre el qual s'ha treballat és un Debian Etch estable (Linux), el compilador GCC, depurador de memòria Valgrind, editor de codi VIM, editor de text LaTeX Kile i gestor de projectes OpenProj. Per tant, no s'ha produït cap cost de software.

Pel que fa al cost del hardware, s'ha emprat un clúster, que comptant les hores que s'ha utilitzat, ha costat 4583'33€ i un ordinador portatil, el preu del qual és 960€.

6.2.3 Cost total

Un cop calculats tots els costos del projecte es pot dir que el cost total del projecte és de:

$$Cost\ Total = 63.240\ e + 4.583'33\ e + 960\ e = 68.783'33\ e$$

Es pot, doncs, classificar el projecte, almenys pel que fa la vessant econòmica com un projecte mitjanament gran.

Capítol 7

Conclusions i treball futur

Aquest capítol té com a objectius per una banda valorar la tasca que s'ha dut a terme, extreure'n les conclusions finals i analitzar cap a quina banda pot tendir en un futur la feina realitzada

7.1 Conclusions

Amb aquest projecte s'ha demostrat que mitjançant l'aplicació de la paral·lelització distribuïda, amb un clúster format per ordinadors de sobretaula, és possible accelerar el procés de Record Linkage. Això cobreix les motivacions que havien dut a iniciar aquest projecte ja que permet que aquelles organitzacions que no disposen de gran recursos tecnològics però si que necessiten aplicar el procés de Record Linkage d'una manera ràpida, com poden ser els centres d'estudis estadístics, puguin aprofitar aquells ordinadors de sobretaula per configurar clústers que els permetin emprar la versió distribuïda del procés de Record Linkage.

El fet d'haver realitzat la paral·lelització tenint en compte les tècniques que existeixen per accelerar la versió seqüencial del procés, és a dir, mètodes de blocking i memoització, aconsegueix que el Record Linkage paral·lel que es proposa no només sigui ràpid pel fet de la paral·lelització sinó que també dóna qualitat des del punt de vista de les garanties d'estar treballant sobre un procés seqüencial que, comparat amb el procés de Record Linkage descrit en les Seccions 2.1 i 2.2, és ràpid.

Cal destacar que s'han fet diferents modificacions de la paral·lelització base. Depenent de la mida del conjunt de dades amb el que s'estigui tractant s'emprarà una de les modificacions o una altra. Si es treballa amb conjunts de registres petits s'usarà la versió Mestre-Treballador, si es treballa amb conjunts de registres mitjanament grans, però que caben en la memòria d'un únic node s'usarà la versió base de la paral·lelització, en la que el Mestre només fa de Mestre, si es treballa amb grans quantitats de registres, que no hi caben en la memòria d'un únic node, es treballarà amb la versió de la paral·lelització en la que els registres estan distribuïts. Les diferents modificacions del procés també apliquen a la configuració del clúster amb la que s'està treballant. Si els nodes que conformen el clúster són ordinadors multi-core es podrà activar la paral·lelització a nivell de node, això permetrà obtenir millor rendiment, és a dir, executar el procés de Record Linkage en menys temps.

7.2 Valoració objectius

Donat que en la Secció 1.3 s'havien fixat uns objectius molt clars resulta interessant emprar aquest objectius per valorar globalment el projecte. Així doncs, en aquesta secció es valorarà

cadascun dels objectius específics.

1 Accelerar el procés de Record Linkage mitjançant tècniques de computació paral·lela.

1.1 *Aprendre a utilitzar MPI.*
Objectiu assolit. Ha calgut aprendre a utilitzar la interfície MPI per tal d'assolir la resta d'objectius ja que és la interfície que s'usa per la comunicació entre els nodes del clúster.

1.2 *Implementar una paral·lelització distribuïda.*
Objectiu assolit. S'ha desenvolupat una versió base de la paral·lelització distribuïda que tenia com a objectiu fer el procés de Record Linkage tan ràpid com fos possible.

1.3 *Utilitzar ordinadors comuns, de sobretaula.*
Objectiu assolit. En el Capítol 5 s'ha demostrat que la configuració d'un clúster composat per ordinadors de sobretaula dóna bons resultats, permeten així implementar el procés de Record Linkage distribuït en aquelles organitzacions que no tenen supercomputadors.

1.4 *Implementar una paral·lelització de memòria compartida a nivell de node.*
Objectiu assolit. S'ha implementat una modificació de la paral·lelització distribuïda base que aprofita el fet que els ordinadors que configuren el clúster tinguin múltiple cores. En el Capítol 5 s'ha vist com aquesta modificació de la paral·lelització base permetia obtenir speed-ups més alts.

1.5 *Dur a terme una implementació conscient de les restriccions de memòria.*
Objectiu assolit. S'ha implementat una modificació de la paral·lelització distribuïda base que aprofita la memòria de tots els ordinadors que composen el clúster en el que s'executa el procés de Record Linkage. Gràcies a aquesta implementació es poden treballar amb conjunts de registres més grans que no caben en la memòria d'un únic node.

1.6 *Utilitzar els mètodes que s'usen per accelerar el RL seqüencial.*

1.6.1 *Entendre el funcionament de les tècniques.*
Objectiu assolit. S'ha fet recerca per conèixer aquelles tècniques que s'empren en la actualitat per accelerar el procés de Record Linkage. S'han estudiat els mètodes de blocking: Standard Blocking i Sliding Window; i les tècniques de memoització.

1.6.2 *Analitzar el que suposa paral·lelitzar aquestes tècniques.*
Objectiu assolit. A partir de l'anàlisi de les tècniques emprades actualment per accelerar el procés de Record Linkage s'ha decidit paral·lelitzar un procés de RL que empra Sliding Window i memoització.

2 *Publicar un article.*

2.1 *Aprendre a escriure un article per a una conferència.*
Objectiu assolit. Si bé és cert que no és pot dir que ara sigui un expert en la redacció d'articles, sí que és pot dir que l'elaboració de l'article "Parallelizing Record Linkage for Disclosure Risk Assessment" va servir per aprendre molt. Cal, però, seguir aprenent per un major assoliment d'aquest objectiu.

2.2 *Explicar en un article aquells punt de recerca que poden tenir un interès en el camp de la recerca.*
Objectiu assolit. En l'article que es va presentar en el PSD-2008 es van explicar

aquells punts del projecte que estaven desenvolupats fins al moment: la paral·lelització base, la paral·lelització a nivell de node i la paral·lelització Mestre-Treballador.

2.3 *Aprendre a presentar un article de recerca amb audiència.*
Objectiu assolit. Malgrat que és una tasca en que l'aprenentatge és continu i l'experiència ajuda a millorar, es valora que la presentació de l'article en el PSD-2008 va anar molt bé. Va ser molt clara i va agradar.

3 *Documentar el Projecte Final de carrera.*

3.1 *Escriure una memòria clara i entenedora que permeti entendre fàcilment la feina feta en el decurs del projecte.*
Malgrat que és molt difícil per a un mateix valorar aquest objectiu, és pot dir que s'ha fet un esforç molt gran en generar un document que fos comprensible sense necessitat de tenir uns coneixements previs de la matèria. S'ha provat de ser clar i anar al gra per motivar al lector a continuar la lectura.

3.2 *Fer una presentació del projecte clara, conscisa i entenedora.*
La valoració és la mateixa que la del punt l'objectiu anterior.

7.3 Treball futur

En aquesta secció s'analitzarà el treball que seria interessant realitzar de cara a l'ampliació del projecte.

En primer lloc caldria integrar entre elles les diferents modificacions de la paral·lelització. Seria molt interessant analitzar, per exemple, com afecta la paral·lelització a nivell de node en el cas de tenir el conjunt de registes distribuïts.

Una altra tasca important que quedaria pendent de realitzar consisteix a trobar una manera d'aprofitar la compartició entre els Esclaus del contingut de les Comparison Stores. S'ha vist que compartir el resultat de les comparacions emmagatzemades en les Comparison Stores és interessant en alguns casos però l'overhead que això causa fa que al final no serveixi absolutament per res, cal doncs buscar d'altres maneres que optimitzin el procés de compartició.

Una altra feina que seria interessant de realitzar ve de l'afirmació de la Secció 4.1 "Una màxima en el món de la computació paral·lela és el balanceig de càrrega." La nostra divisió ha suposat un clúster homogeni, és a dir, un clúster en que tots els computadors que el formen tenen les mateixes característiques. Sota aquesta suposició la nostra divisió de la feina en S Sfinestres, on cada Sfinestra és de mida $N/S+(w-1)$[1] on N és el número total de registres i S el número d'Esclaus, és correcta. Però què passaria si el clúster fos heterogeni? És a dir, què passa si els diferents ordinadors que conformen el clúster tenen característiques completament diferent entre ells? Això d'entrada no és quelcom recomanable, però és interessant analitzar què passaria en aquest context si s'executés la paral·lelització proposada. El cas més clar és per exemple, què passaria si un dels Esclaus és un node que té molta menys memòria que la resta? Doncs les seves Comparison Stores seran molt més petites i, per tant haurà de fer molt més càlcul. O què passaria si un dels Esclaus tingués un processador molt més lent? Doncs que pel mateix nombre de comparacions trigaria molt més que un altre Esclau amb un processador més ràpid. En ambdos casos s'estaria desaprofitant el veritable potencial del clúster. Dit això, seria interessant desenvolupar un algoritme de divisió de la feina que per

[1]Totes les Sfinestres excepte la primera, que és de mida N/S i la última que és $N/S+(w-1)+r$, són de mida $N/S+(w-1)$

balancejar la càrrega que tingués en compte, entre d'altres, aquests dos factors, la memòria i el processador, per fer la divisió en Sfinestres. Notis aquest algoritme permetria, també, treure tot el potencial a la modificació Mestre-Treballador.

Per últim, una modificació interessant a tenir en compte en la versió de la paral·lelització en la que els registres estan distribuïts és millorar l'algoritme d'ordenació distribuïda i emprar-ne algun dels que existeixen en la literatura i tenen bons resultats, caldria, però, analitzar exactament l'impacte que això té en el codi de DAURUM.

7.4 Valoració personal

Donat que aquest projecte s'engloba dins del marc del projecte final de carrera, i com a tal suposa el final d'una etapa important, resulta interessant fer una valoració més de caire personal del que ha suposat dur a terme aquest projecte.

El desenvolupament del projecte m'ha permés d'alguna manera introduir-me en el món laboral, donat que el projecte s'ha desenvolupat en el si del grup de recerca DAMA-UPC. Val a dir que realitzar el projecte en el grup ha estat un veritable plaer. Des del primer moment m'hi vaig sentir molt ben acollit i això ha permés realitzar la meva feina d'una manera molt més còmode. Estar envoltat de persones amb una amplitud de coneixements tan gran com la que hi ha en el grup sempre ajuda a aprendre i per tant a millorar la qualitat de la feina realitzada.

M'ha ajudat a clarificar les meves properes passes un cop finalitzada la carrera doncs la intenció és seguir en el món de la recerca, i això és quelcom que abans de començar el projecte no m'havia ni tan sols plantejat. No puc deixar d'explicar que l'acceptació d'un article en una conferència i la seva posterior presentació en públic ha estat quelcom molt positiu en la meva formació.

Per altra banda, l'execució d'aquest projecte m'ha ajudat a consolidar els coneixements adquirits a través dels meus estudis en les diferents assignatures de la carrera. Està clar que ha calgut aplicar molts dels coneixements adquirits en les assignatures obligatòries, i d'altres optatives com **Multiprogramació**, **Programació Concurrent i Distribuïda**, **Sistemes Operatius Distribuïts i en Xarxa** o **Configuració i Avaluació del rendiment del Sistemes.**

En general em sento força satisfet amb la feina aconseguida amb aquest projecte, per una banda per l'alt grau d'assoliment d'objectius i per totes aquestes qüestions de caire més personal.

Apèndix A

Estadistica dels Noms i Cognoms de Catalunya

A.1 Noms

Nom	Posició total homes i dones	Freqüència	tant per mil sobre total homes i dones
JOSÉ	1	126389	17,51
ANTONIO	2	121122	16,78
MARIA/MARÍA	3	119429	16,55
FRANCISCO	4	95264	13,2
MONTSERRAT	5	89178	12,36
JUAN	6	85460	11,84
MANUEL	7	84948	11,77
MARÍA DEL CARMEN	8	75219	10,42
JORDI	9	74243	10,29
CARMEN	10	73426	10,17
DAVID	11	64979	9
JOSEFA	12	60516	8,38
MARTA	13	51333	7,11
NÚRIA/NURIA	14	51270	7,1
JOSEP	15	50394	6,98
ISABEL	16	48854	6,77
JOAN	17	48006	6,65
CARLOS	18	46097	6,39
LAURA	19	46052	6,38
RAMON/RAMÓN	20	45883	6,36
MARIA TERESA MARÍA TERESA	21	45384	6,29
DANIEL	22	44310	6,14
ANTÒNIA/ANTONIA	23	43847	6,08
DOLORES	24	43248	5,99
FRANCISCA	25	43212	5,99

CRISTINA	26	42766	5,93
MARC	27	41618	5,77
PEDRO	28	40401	5,6
ROSA	29	39671	5,5
ANA	30	39484	5,47
JAVIER	31	38166	5,29
JOSÉ MARÍA	32	38098	5,28
MIGUEL	33	37870	5,25
ANA MARÍA	34	37406	5,18
TERESA	35	36110	5
JORGE	36	35843	4,97
PILAR	37	33883	4,69
MARÍA DOLORES	38	33725	4,67
RAFAEL	39	32747	4,54
ANNA	40	32605	4,52
MERCEDES	41	31300	4,34
ALBERT	42	30560	4,23
LUIS	43	30382	4,21
JOSÉ ANTONIO	44	30297	4,2
XAVIER	45	29087	4,03
SÍLVIA/SILVIA	46	28764	3,99
ÀNGEL/ÁNGEL	47	28364	3,93
FRANCISCO JAVIER	48	28256	3,91
JOSÉ LUIS	49	27708	3,84
MARÍA DEL PILAR	50	27226	3,77
SERGIO	51	27065	3,75
ROSA MARIA ROSA MARÍA	52	26831	3,72
CONCEPCIÓN	53	25575	3,54
ÒSCAR/ÓSCAR	54	25220	3,49
ALEJANDRO	55	24531	3,4
JUANA	56	23982	3,32
JESÚS	57	23825	3,3
ALBERTO	58	23472	3,25
MARIA ISABELR MARÍA ISABEL	59	23385	3,24
JAUME	60	23236	3,22
MÒNICA/MÓNICA	61	22886	3,17
MARIA ROSA MARÍA ROSA	62	22789	3,16
FERNANDO	63	22333	3,09
JOAQUÍN	64	22292	3,09
JAIME	65	22112	3,06
MARÍA ÁNGELES	66	21971	3,04
EVA	67	21399	2,96

SANDRA	68	20896	2,9
ROSARIO	69	20516	2,84
SÒNIA/SONIA	70	20190	2,8
VÍCTOR	71	20030	2,78
ENRIQUE	72	19804	2,74
RAQUEL	73	19736	2,73
MANUELA	74	19639	2,72
SARA	75	19431	2,69
JÚLIA/JULIA	76	19300	2,67
MIREIA	77	18895	2,62
RAÜL/RAÚL	78	18705	2,59
IVAN/IVÁN	79	18683	2,59
JOSEP MARIA	80	18433	2,55
SALVADOR	81	18242	2,53
ESTHER	82	17924	2,48
MIGUEL ÁNGEL	83	17720	2,46
FRANCESC	84	17611	2,44
YOLANDA	85	17463	2,42
JUAN JOSÉ	86	17340	2,4
MOHAMED	87	17335	2,4
GEMMA	88	17307	2,4
ELENA	89	17292	2,4
ALBA	90	17289	2,4
MARÍA JOSÉ	91	17193	2,38
SUSANA	92	16943	2,35
MARÍA LUISA	93	16909	2,34
SERGI	94	16716	2,32
JOSÉ MANUEL	95	16621	2,3
LAIA	96	16579	2,3
MARINA	97	16571	2,3
MARGARITA	98	16561	2,29
SANTIAGO	99	16476	2,28
ENCARNACIÓN	100	16311	2,26
Suma parcial del %:			48,91
PERE	101	16046	2,22
JUAN CARLOS	102	16038	2,22
PAULA	103	15881	2,2
ANDRÉS	104	15561	2,16
MIQUEL	105	15532	2,15
JUAN ANTONIO	106	15061	2,09
LLUÍS	107	14925	2,07
ÀLEX/ÁLEX	108	14285	1,98
ANDREA	109	14266	1,98
OLGA	110	13966	1,94
VICENTE	111	13842	1,92

RUBÈN/RUBÉN	112	13794	1,91
EDUARDO	113	13752	1,91
ORIOL	114	13735	1,9
GERARD	115	13641	1,89
PAU	116	13517	1,87
LÍDIA/LIDIA	117	13302	1,84
ÀNGELA/ÁNGELA	118	13108	1,82
ANTONI	119	13063	1,81
MÍRIAM/MIRIAM	120	13052	1,81
CARLA	121	12988	1,8
IRENE	122	12912	1,79
CARLES	123	12480	1,73
RICARDO	124	12428	1,72
JOSEFINA	125	12220	1,69
MERCÈ	126	11932	1,65
JUAN MANUEL	127	11893	1,65
NATÀLIA/NATALIA	128	11855	1,64
GLÒRIA/GLORIA	129	11690	1,62
ÁNGELES	130	11640	1,61
PATRÍCIA/PATRICIA	131	11430	1,58
DIEGO	132	11419	1,58
EMILIO	133	11193	1,55
PABLO	134	11069	1,53
ALFONSO	135	10997	1,52
POL	136	10982	1,52
ENRIC	137	10910	1,51
ROGER	138	10905	1,51
ELISABET	139	10734	1,49
JUDIT	140	10729	1,49
CARME	141	10481	1,45
ALÍCIA/ALICIA	142	10469	1,45
ADRIÁN	143	10276	1,42
CRÍSTIAN CRISTIAN CRISTIÁN	144	10228	1,42
ARNAU	145	10167	1,41
CLÀUDIA/CLAUDIA	146	10156	1,41
AGUSTÍN	147	10033	1,39
ADRIÀ	148	9981	1,38
MARIA JESÚS MARÍA JESÚS	149	9967	1,38
LUCÍA	150	9926	1,38
LOURDES	151	9905	1,37
MAGDALENA	152	9881	1,37
MARIA DOLORS	153	9613	1,33

MARÍA JOSEFA	154	9611	1,33
ROSER	155	9557	1,32
TOMÀS/TOMÁS	156	9541	1,32
CAROLINA	157	9504	1,32
EMÍLIA/EMILIA	158	9470	1,31
EDUARD	159	9283	1,29
ESTER	160	9232	1,28
ERIC	161	9209	1,28
GABRIEL	162	9187	1,27
LUISA	163	9063	1,26
JUDITH	164	9061	1,26
MARIA ANTÒNIA MARÍA ANTONIA	165	9052	1,25
FERRAN	166	9036	1,25
MERITXELL	167	8968	1,24
MARIA DEL CARME	168	8909	1,23
DOLORS	169	8894	1,23
VICTÒRIA/VICTORIA	170	8882	1,23
ARIADNA	171	8869	1,23
JOAQUIM	172	8836	1,22
VERÒNICA/VERÓNICA	173	8762	1,21
MARÍA MERCEDES	174	8618	1,19
RAMONA	175	8439	1,17
MARCOS	176	8388	1,16
BEATRIZ	177	8080	1,12
CLARA	178	8031	1,11
JULIÁN	179	7996	1,11
CATALINA	180	7957	1,1
GUILLEM	181	7945	1,1
JOEL	182	7804	1,08
IGNACIO	183	7715	1,07
MARIO	184	7714	1,07
MARIA DEL MAR MARÍA DEL MAR	185	7608	1,05
INMACULADA	186	7578	1,05
DOMINGO	187	7543	1,05
NOÈLIA/NOELIA	188	7539	1,04
ANNA MARIA	189	7470	1,03
AURORA	190	7420	1,03
VANESSA	191	7405	1,03
CONSUELO	192	7392	1,02
MARIA MONTSERRAT MARÍA MONTSERRAT	193	7355	1,02
JULIO	194	7223	1
FRANCISCO JOSÉ	195	7097	0,98

FÈLIX/FÉLIX	196	7044	0,98
SEBASTIÁN	197	7025	0,97
JESSICA	198	6976	0,97
FÀTIMA/FÁTIMA	199	6921	0,96
AMPARO	200	6846	0,95
MARTÍ	201	6805	0,94
ROBERTO	202	6751	0,94
FRANCESC XAVIER	203	6740	0,93
TRINIDAD	203	6740	0,93
ASUNCIÓN	205	6703	0,93
ALEIX	206	6501	0,9
AHMED	207	6361	0,88
LORENA	208	6358	0,88
EVA MARIA/EVA MARÍA	209	6330	0,88
NOEMÍ	210	6263	0,87
GUILLERMO	211	6251	0,87
BERTA	212	6168	0,85
NEUS	213	6137	0,85
INÉS	214	6097	0,84
ELVIRA	215	6080	0,84
RICARD	216	6071	0,84
ISMAEL	217	6018	0,83
MATILDE	218	5992	0,83
ESTEBAN	219	5964	0,83
MARIANO	220	5956	0,83
ARACELI	221	5901	0,82
MARIA ELENA MARÍA ELENA	222	5898	0,82
MARIA ÀNGELS	223	5887	0,82
JOAQUINA	224	5870	0,81
ESPERANZA	225	5667	0,79
REMEDIOS	226	5639	0,78
JOANA	227	5613	0,78
NATIVIDAD	228	5598	0,78
MARIA VICTÒRIA MARÍA VICTORIA	229	5556	0,77
MARÍA ASUNCIÓN	230	5503	0,76
MANEL	231	5482	0,76
EULÀLIA/EULALIA	232	5468	0,76
MARTÍN	233	5393	0,75
ENRIQUETA	234	5377	0,74
ISAAC	235	5366	0,74
NEREA	236	5361	0,74
PURIFICACIÓN	236	5361	0,74
ALFREDO	238	5339	0,74

ÁLVARO	239	5330	0,74
ELISABETH	240	5253	0,73
FELIPE	241	5216	0,72
MOHAMMED	242	5165	0,72
CRISTÓBAL	243	5109	0,71
NIEVES	244	5103	0,71
HÈCTOR/HÉCTOR	245	5075	0,7
VANESA	246	5042	0,7
RAFAELA	247	4979	0,69
MARIA CRISTINA MARÍA CRISTINA	248	4957	0,69
ELISA	249	4946	0,69
ANDREU	250	4944	0,68
CONCEPCIÓ	251	4919	0,68
HELENA	252	4847	0,67
SOLEDAD	253	4820	0,67
ADELA	254	4808	0,67
AÏDA/AIDA	255	4806	0,67
AINA	256	4758	0,66
NICOLÁS	257	4665	0,65
JUAN FRANCISCO	258	4647	0,64
MARÍA LOURDES	259	4645	0,64
TÀNIA/TANIA	260	4602	0,64
JOSÉ RAMÓN	261	4601	0,64
LORENZO	262	4571	0,63
ALEXANDRE	263	4564	0,63
ANA ISABEL	264	4497	0,62
JOSÉ MIGUEL	264	4497	0,62
JONATHAN	266	4482	0,62
CÈLIA/CELIA	267	4458	0,62
MARIA CINTA MARÍA CINTA	268	4440	0,62
ISIDRO	269	4430	0,61
ENCARNACIÓ	270	4426	0,61
MARÍA ROSARIO	271	4424	0,61
ESTEFANIA/ESTEFANÍA	272	4406	0,61
EMMA	273	4373	0,61
LEONOR	274	4338	0,6
MARIONA	275	4304	0,6
AITOR	276	4301	0,6
ROCÍO	277	4259	0,59
VICENTA	278	4231	0,59
GREGORIO	279	4201	0,58
MARIA MERCÈ	280	4136	0,57
BLANCA	281	4130	0,57

ABEL	282	4117	0,57
MILAGROS	283	4097	0,57
ASCENSIÓN	284	4023	0,56
MARÍA NIEVES	285	3983	0,55
AMÀLIA/AMALIA	286	3938	0,55
CÈSAR/CÉSAR	287	3897	0,54
IGNASI	288	3866	0,54
JONATAN	289	3849	0,53
SOFIA/SOFÍA	290	3816	0,53
ROSALIA/ROSALÍA	291	3812	0,53
FELISA	292	3760	0,52
EDGAR	293	3759	0,52
ROBERT	294	3739	0,52
ADRIANA	295	3714	0,51
MARÍA CONCEPCIÓN	296	3712	0,51
ALEXANDRA	297	3683	0,51
CHRISTIAN	298	3680	0,51
MARÍA SOLEDAD	299	3659	0,51
AINHOA	300	3658	0,51
Suma total del %:			70,22

S'observa que amb els primers cent noms representen el nom de gairebé el 50% de la població catalana. I que els els primers tres-cents noms representen més del 70% dels noms de la població catalana. Això reforça el que s'ha explicat en el Capítol 3.

A.2 Cognoms

Cognom	Posició Catalunya	Freqüència com a 1r cognom	Tant per mil	Freqüència com a 2n cognom	Tant per mil
GARCIA	1	169026	23,42	172071	23,84
MARTINEZ	2	117944	16,34	118963	16,48
LOPEZ	3	112845	15,63	115414	15,99
SANCHEZ	4	101557	14,07	103251	14,31
RODRIGUEZ	5	97518	13,51	98034	13,58
FERNANDEZ	6	96308	13,34	98093	13,59
PEREZ	7	91366	12,66	93675	12,98
GONZALEZ	8	89854	12,45	90303	12,51
GOMEZ	9	55659	7,71	56178	7,78
RUIZ	10	49629	6,88	51574	7,15
MARTIN	11	45602	6,32	44896	6,22
JIMENEZ	12	43267	5,99	42438	5,88
MORENO	13	41620	5,77	41876	5,8
MUÑOZ	14	37754	5,23	38083	5,28
HERNANDEZ	15	36999	5,13	37502	5,2
DIAZ	16	33886	4,69	33882	4,69
NAVARRO	17	28309	3,92	28565	3,96
ROMERO	18	28206	3,91	27931	3,87
ALVAREZ	19	26079	3,61	26103	3,62
TORRES	20	26150	3,62	25984	3,6
MOLINA	21	19389	2,69	19657	2,72
GUTIERREZ	22	19536	2,71	19296	2,67
SERRANO	23	18633	2,58	18639	2,58
VILA	24	18411	2,55	17845	2,47
GIL	25	18178	2,52	18074	2,5
VIDAL	26	17205	2,38	16974	2,35
RAMIREZ	27	17113	2,37	17047	2,36
MORALES	28	16320	2,26	16313	2,26
RAMOS	29	16482	2,28	16101	2,23
SERRA	30	16691	2,31	15723	2,18
MARTI	31	16616	2,3	15751	2,18
MARIN	32	16257	2,25	15760	2,18
GIMENEZ	33	15182	2,1	16222	2,25
ORTIZ	34	15369	2,13	15821	2,19
FERRER	35	15850	2,2	15268	2,12
ORTEGA	36	15328	2,12	15506	2,15
SOLER	37	15396	2,13	15272	2,12
ALONSO	38	15246	2,11	15334	2,12
DOMINGUEZ	39	14650	2,03	14736	2,04
RUBIO	40	14669	2,03	14659	2,03
PUIG	41	14744	2,04	14136	1,96

DELGADO	42	13930	1,93	13843	1,92
ROCA	43	13715	1,9	13836	1,92
VAZQUEZ	44	13706	1,9	13625	1,89
CORTES	45	13251	1,84	13092	1,81
CASTRO	46	12830	1,78	13087	1,81
CASTILLO	47	12766	1,77	13074	1,81
CANO	48	12182	1,69	12227	1,69
SOLE	49	11945	1,65	12366	1,71
FLORES	50	11673	1,62	12212	1,69
PUJOL	51	11459	1,59	11494	1,59
BLANCO	52	11258	1,56	11103	1,54
GUERRERO	53	11166	1,55	11189	1,55
FONT	54	11461	1,59	10811	1,5
MEDINA	55	10846	1,5	11162	1,55
DURAN	56	10975	1,52	10719	1,49
LOZANO	57	10692	1,48	10849	1,5
PASCUAL	58	10532	1,46	10322	1,43
AGUILAR	59	10352	1,43	10198	1,41
NUÑEZ	60	10215	1,42	10312	1,43
COSTA	61	10332	1,43	10046	1,39
GARRIDO	62	10002	1,39	10115	1,4
CRUZ	63	9982	1,38	9984	1,38
CASAS	64	10176	1,41	9515	1,32
MARQUEZ	65	9676	1,34	9541	1,32
SEGURA	66	9517	1,32	9275	1,29
ROVIRA	67	9598	1,33	9113	1,26
SALA	68	9418	1,3	9046	1,25
PONS	69	9259	1,28	9152	1,27
CARMONA	70	9156	1,27	9180	1,27
SANZ	71	9078	1,26	8897	1,23
BOSCH	72	9142	1,27	8636	1,2
GALLEGO	73	8965	1,24	8715	1,21
FUENTES	74	8874	1,23	8789	1,22
MENDEZ	75	8523	1,18	8689	1,2
CAMPOS	76	8461	1,17	8582	1,19
CABALLERO	77	8415	1,17	8621	1,19
CARRASCO	78	8662	1,2	8274	1,15
MAS	79	8684	1,2	8134	1,13
ROIG	80	8521	1,18	8206	1,14
LEON	81	8202	1,14	8510	1,18
HIDALGO	82	8267	1,15	8339	1,16
MORA	83	8249	1,14	8087	1,12
MOYA	84	8011	1,11	7972	1,1
SOLA	85	7993	1,11	7900	1,09
RIERA	86	8169	1,13	7709	1,07
IBAÑEZ	87	7759	1,08	7950	1,1

CALVO	88	7848	1,09	7755	1,07
GRAU	89	8021	1,11	7473	1,04
REYES	90	7792	1,08	7591	1,05
IGLESIAS	91	7703	1,07	7675	1,06
PEÑA	92	7537	1,04	7782	1,08
HERRERA	93	7629	1,06	7637	1,06
SANTOS	94	7553	1,05	7696	1,07
TOMAS	95	7514	1,04	7559	1,05
DOMENECH	96	7632	1,06	7386	1,02
GALLARDO	97	7346	1,02	7475	1,04
VALLS	98	7719	1,07	7045	0,98
PARRA	99	7038	0,98	7415	1,03
LUQUE	100	6898	0,96	7171	0,99
Suma total del %:			31,05		31,16

Apèndix B

Article presentat en el congrés PSD2008

"Privacy in Statistical Databases 2008"(PSD 2008) és un congrés patrocinat i organitzat per la Càtedra UNESCO en Privacitat de dades. Els articles publicats estan publicats per Springer-Verlag en la secció d'Informàtica.

La privacitat en les bases de dades estadístiques té com a objectiu trobar l'equilibri entre la demanada, cada cop més gran, de la societat i dels agent econòmics per obtenir informació acurada i la legalitat, o obligacions ètiques, de protegir la privacitat de les persones i organitzacions de les que es vol extreure la informació.

La privacitat és un repte per les agències estadístiques i/o els centres d'estudi no poden esperar aconseguir informació acurada de les persones o organitzacions sense que aquestes tinguin la completa seguretat que la seva seguretat està garantida.

Cal doncs desenvolupar mètodes que siguin capaços de fer anònimes les bases de dades conservant, però, les propietats estadístiques de la informació que està emmagatzemada. Cal però que aquestes mètodes d'anonimització siguin segurs, és a dir, que no es pugui relacions la informació que conté la base de dades anonimitzada amb les entitats del món real. És aquí on el Record Linkage juga un paper important. El Record Linkage esdevé una bona tècnica per mesurar la qualitat dels mètodes d'anonimització. Si donada una base de dades, i la mateixa base de dades anonimitzada, si el procés de Record Linkage és capaç de relacionar els registres de la base de dades anonimitzada amb la entitat real corresponent en la base de dades no anonimitzada, vol dir que, el mètode d'anonimització proposat és de baixa qualitat. Quantes més pares sigui capaç de relacionar el procés de RL de menys qualitat serà la tècnica d'anonimització de les dades proposades.

L'article es va presentar justificant la necessitat de trobar maneres d'accelerar el procés de Record Linkage, i es proposava per fer-ho una paral·lelització distribuïda. L'article es va acceptar i es va presentar el 25 de Setembre a Istambul (Turquia). L'article va tenir molt bona acceptació entre els assistents al congrés.

Les següent pàgines es corresponen a l'article que es va presentar en la conferència Privacy in Statistical Database.

Parallelizing Record Linkage for Disclosure Risk Assessment

Joan Guisado-Gámez[1], Arnau Prat-Pérez[1], Jordi Nin[2], Victor Muntés-Mulero[1], and Josep Ll. Larriba-Pey[1]

[1]DAMA-UPC, Dept. d'Arquitectura de Computadors
Universitat Politècnica de Catalunya,
Campus Nord, C/Jordi Girona 1-3
08034 Barcelona, Catalonia, Spain
{joan,aprat,vmuntes,larri}@ac.upc.edu
http://www.dama.upc.edu/
[2]IIIA, Artificial Intelligence Research Institute
CSIC, Spanish National Research Council
Campus UAB s/n
08193 Bellaterra Catalonia, Spain
jnin@iiia.csic.es

Abstract. Handling very large volumes of confidential data is becoming a common practice in many organizations such as statistical agencies. This calls for the use of protection methods that have to be validated in terms of the quality they provide. With the use of Record Linkage (RL) it is possible to compute the disclosure risk, which gives a measure of the quality of a data protection method. However, the RL methods proposed in the literature are computationally costly, which poses difficulties when frequent RL processes have to be executed on large data.

Here, we propose a distributed computing technique to improve the performance of a RL process. We show that our technique not only improves the computing time of a RL process significantly, but it is also scalable in a distributed environment. Also, we show that distributed computation can be complemented with SMP based parallelization in each node increasing the final speedup.

Keywords: Record linkage, parallel computing, distributed computing, disclosure risk evaluation.

1 Introduction

The need for data protection methods is larger every day, becoming crucial to anonymize confidential information before releasing it in a private manner. This is true in many situations where data becomes public or semi-public, and a corrupt use of it may lead to the disclosure of such confidential information. A situation where this may arise is when data is released by statistical agencies, where there is a need to preserve the statistical properties of the information while keeping it anonymous.

J. Domingo-Ferrer and Y. Saygin (Eds.): PSD 2008, LNCS 5262, pp. 190–202, 2008.

However, when a protection method is applied, the evaluation of the privacy provided by such method becomes a problem. Re-identification techniques, such as Record Linkage (RL) methods [16,17,18], are the most common techniques for evaluating the quality of a given protection method, *i.e.* the disclosure risk. RL methods model the situation where an intruder sees the protected recordset whereas he has access to records of the original recordset obtained from other sources. The goal of the RL methods used by an intruder is to link the original records with the corresponding records in the protected recordset. As a consequence, the larger the number of records linked by means of these record linkage methods, the larger the disclosure risk of the protection method.

Nowadays, there are two important aspects in the anonymization process. First, the amount of data collected is larger every day due to the availability of larger population databases. Second, the need for faster methods is also more important because it is necessary to provide service to more frequent demands. These two aspects call for the use of parallel applications during the RL process.

In this paper, we propose a distributed strategy that focusses on speeding up the RL processes for large data volumes. We propose this assuming that the recordset fits in the memory of a single computer, deploying a strategy to perform the most expensive part of the whole RL computation on a farm of slave computers with independent memories. The simplicity of the method makes it viable on a set of personal computers connected through a LAN, which makes the whole system feasible for situations where there is a need for speed but the resources for large parallel computers are not available.

Our results show that the use of parallel computing devices on large data sets improves the performance of the RL methods leading the larger sets of data on large numbers of processors to linear speedups of almost P for P processors. The results also show that the overhead of performing a distributed execution of the RL code does not grow with the number of processors, demonstrating the scalability of the strategies proposed. Note also that, having multicore processors make it possible to implement an SMP based parallelization in each node in order to achieve a better performance.

The rest of the paper is organized as follows. We start setting up the problem and describing a memoization technique in Section 2. We describe the technique to distribute the computations in Section 3. We evaluate the parallelization in Section 4. Finally, we explain the related work in Section 5 and conclude in Section 6.

2 Record Linkage

Record Linkage aims at processing a set of recordsets in order to obtain sets of records that belong to the same unique individual. We consider that RL is formed by different phases as shown in Figure 1. First, the data sources are cleaned and pre-processed normalizing attributes in the recordset files individually to allow a simpler comparison with other data in the following steps [9].

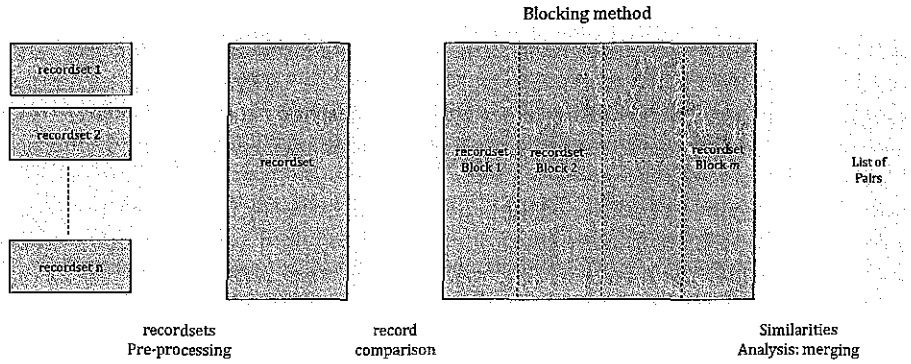

Fig. 1. Record Linkage processing model

Once the pre-processing is done, RL proceeds with the record comparison. The objective of this phase is to obtain pairs of records that possibly belong to the same individual. There are two kinds of RL algorithms for record comparison: those based on probabilistic methods and those based on distance functions [16,17]. During the RL process, records are compared following a strategy that may have several objectives, like reducing the number of comparisons as with the Standard Blocking[2,7] or Sliding Window [10] (also known as Sorted Neighborhood) methods, or finding the largest groups of similar records at the lowest comparison cost as with Reduction using Anchor Record (RAR) [14].

In order to avoid possible errors induced by blocking methods, it is usual to perform several passes using different sorting criteria, like the name or the surname in the case that the entities are human beings.

Finally, it is necessary to analyze the record pairs so that groups of similar records are formed and false positives are discarded if possible, before the eventual expert review process is done.

Record Comparison Process

The comparison phase is the most expensive in the RL process, with quadratic complexity ($O(N^2)$) in the number of records N, as opposed to the linear complexity of the other phases. In this paper we will use the Sliding Window method in order to reduce the record comparison phase complexity to ($O(BN)$), where B is the size of the block used (window). However, it is still the most complex phase in the whole RL process. Therefore, several additional techniques have been proposed in order to further improve performance. Among these, the use of memoization techniques for reducing the number of comparisons has been proven as one of the most effective in terms of performance [6]. In this paper, we assume this proposal as the baseline for our work.

After these considerations and before presenting our technique, we briefly describe the different phases of the Record Comparison process used in this paper. The Record Comparison process is divided into four phases: Comparison Preprocess, Caching, Detection and Merging. Following, we describe these phases.

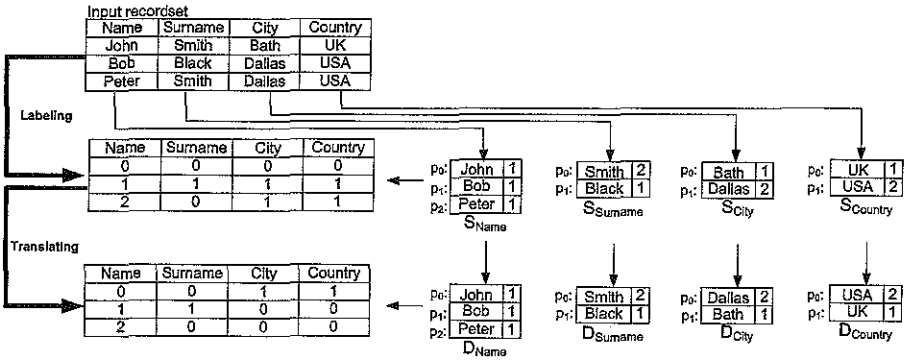

Fig. 2. Labeling and Translating steps

Comparison Preprocess. This phase aims at reducing the comparison cost and making the use of memoization techniques simpler in order to reduce the amount of computations involved in the comparison. We distinguish between two steps in this phase. The first step is **Labeling**, as shown in Figure 2, where the string values in the records are replaced with integer identifiers, and dictionaries are created to match each string with its identifier. This allows to perform exact comparisons between identical strings very efficiently, as well as, preparing the data to make it possible to use memoization techniques with those strings that are not equal.

Although the use of identifiers simplifies the exact comparison of attributes, it is still necessary to compare the non-exact matching values. In our case, this is done by using a two level String Comparison Function based on the Levenshtein distance [15] applied to pairs of tokens coming from the compared string values. Note that the objective of the following steps is to minimize the computational cost of using such approximate comparison function.

As Figure 2 shows, for each comparison attribute A, a list S_A is created at loading time. $S_A = \{p_0, p_1, \cdots, p_{n-1}\}$ where n is the number of unique strings in A. Each p_i is a pair $(v_i, |v_i|)$ where v_i is a string value in A, and $|v_i|$ is its number of occurrences in A. At the same time, each value v in A is replaced by i (*i.e.* its position in the list is used as a string identifier), where $p_i = (v_i, |v_i|)$, $v_i = v$ and $|v_i|$ is the number of occurrences of v in A. Note that the structures on the right hand side of Figure 2 show the dictionaries, while the left hand side part illustrates the data transformation from a string to an identifier.

The second step is **Translating**, also depicted in Figure 2. It consists in sorting S_A obtaining a new structure denoted by D_A. D_A has the same elements than S_A but they have been sorted by the number of times they appear in A. Then, identifiers are reassigned, giving the smallest identifier to the most frequent value. Note that because of the sorting process, it is necessary to translate the identifiers of each value v of A to the new position occupied in D_A.

Memory Allocation. In this phase, a cache for each comparison attribute A is created. Note that, the caches are only created but not populated. These caches

are used during the rest of the phases. The caching structures allow storing the results of the comparisons, minimizing the number of actual comparison computations. Usually, the number of unique values in A makes it unfeasible to store the result for all the possible pairs of strings in memory, thus, only a subset of the comparisons is stored. Given that, the cache size depends on the memory available in the system, only m elements in A can be represented. Since D_A is sorted decreasingly by the number of occurrences, the comparisons between the m most frequent strings in A are stored in the cache. Therefore, the comparison between those frequent values are memoized avoiding unnecessary comparisons during the process.

These caches are called Comparison Stores (CSs), and they are proposed and described in detail in [6].

Detection. In this phase the comparisons between records are performed and finally the similarities are detected. In order to find the maximum number of similarities, several passes of a blocking method are performed. Each pass uses a different criteria for sorting the recordset.

During this phase, the Comparison Stores are populated and used intensively. When the comparison of a frequent pair of strings is performed, the result is stored in the CS and it can be used later when the same strings have to be compared. Thus, this result will never be computed again, minimizing the number of comparisons for the frequent values.

Merging. This phase focusses on creating a similarity list removing duplicated similarities that appear as a consequence of doing several passes in the Detection phase. A similarity pair is only inserted in the list if it has not been inserted previously. In order to achieve a good performance, a hash table is used for controlling the similarities that have been already inserted in the list.

3 Record Comparison Process Parallelization

Now, we describe the parallelization of the Record Comparison process. We take the Record Comparison schema presented in Section 2 as the baseline because it is the most recent high performance RL strategy to our knowledge. However, the parallel technique we present in this paper can be used on any Record Comparison Process.

In this paper, we parallelize the Record Comparison process. Given a set of nodes, each one processes a portion of the recordset. Figure 3 shows the cluster-based parallelization that follows a Master/Slave architecture. The Master node is responsible for maintaining the whole input recordset and for sending the data necessary to each Slave node for processing. In fact, this could have two different implementations. First, assuming that all the nodes have all the data, thus, the parallelization comes from the tasks on different data subsets engaged by the different nodes. Second, assuming that only the Master node stores the data, and the Slaves receive the data subsets necessary at each precise moment. We opted for the

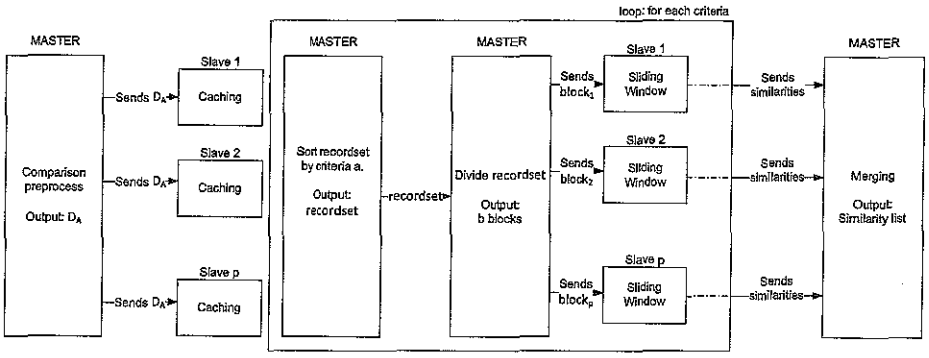

Fig. 3. Parallelization of the Record Comparison Process

second option because it maximizes the amount of space for auxiliary data structures in the Slaves. Thus, by saving record space, we can create larger Comparison Stores, which is significantly beneficial as shown in Section 4.

Assuming that the Master node is the unique node that keeps the whole recordset, it is the only node that can perform the Record Comparison preprocess, and create the D_A lists. As the Slaves need the D_A lists in order to construct their own caches and for the Detection phase, the Master node has to send them to the Slaves. Note that each Slave has its own independent caches (*i.e.* the cached comparisons are not shared among Slaves) so the same comparison may be performed in each Slave, in contrast with the sequential version where the same comparison is performed only once. This part of the preprocess is sequential and it will become the most expensive part of the computations when parallelism is applied, as we show later.

Once all the Slaves have their own D_A structures and caches, they are ready to perform the Detection phase. For each pass in the Detection phase, the Master node has to sort the recordset by the current criteria. Once the recordset is sorted, it is divided into a set of data blocks, in our case $b = \frac{|recordset|}{|Slaves|}$ blocks. The Master node sends each block to a different Slave, and if the Master plays the Slave role, it keeps the last block. At that point, as explained in Section 2, each Slave can start performing the Sliding Window strategy over its block. Once a Slave has finished performing Sliding Window, it has to wait until the Master node sends a new block to be processed. Note that, if the Master plays the Slave role, it will have to finish applying the Sliding Window strategy over its own block before sorting the recordset by the next criteria, dividing it into b blocks and sending them to the Slaves. Therefore, the slower the Master node in performing the Sliding Window, the longer the Slaves will be waiting idle, without working. The convenience of using the Master as a Slave will be discussed in Section 4.

Each Slave at the Detection phase creates a list with the similarities it finds. Because of the several passes that are performed during the Detection phase, some similarities may be duplicated among the lists. Therefore it is necessary that the Slaves send their own list to the Master node. The Master node will merge them creating a similarity list where no duplicated similarities will appear.

4 Experiments

With the objective of evaluating the proposed parallelization, we run a set of experiments that shows the interaction between the size of the recordset with the number of nodes to perform the RL process. To compare the results of our parallelization we will use the Record Comparison process described above which is the best possible sequential algorithm at hand.

The experiments use different numbers of records ranging from 1 million (1M) to 8 million (8M) 128 byte records. The recordsets used in these experiments have been generated using the synthetic record file generator included in the FEBRL toolkit [11], with a 30 percent of duplicates and a maximum of 10 duplicates per record. However, in order to make the whole evaluation as realistic as possible, the frequencies of the names and surnames used to generate the synthetic recordsets with FEBRL have been obtained from the Catalan Statistics Institute [5]. Recordsets are composed of records with eight attributes, out of which four are strings: first name, first surname, second surname, as in the Catalan person-naming system, and address.

In order to perform the experiments we have used a Beowulf [12] cluster with the features described in Table 1.

Table 1. Cluster description

	Beowulf cluster
Number of nodes:	16
Processor of each node:	Intel Core 2 Duo 6600 @2.4Ghz
Memory of each node:	2GB
L2 Cache size of each node:	4096KB
Network:	1Gbps

Comparison of Parallel Strategies

With the objective to minimize the effort to parallelize, we first want to understand the advantages of using the Master as a Slave or not. We will refer to the version where the Master node acts as a Slave, as the Worker-Master version, and the version where it does not, as the Lazy-Master version.

Figure 4 shows the Record Comparison process time in (a), and the cache size of the Master and the Slaves in (b), when different amount of records are considered in both versions. The number of nodes is fixed to 16. Note that, there are 15 Slaves and 1 Master. Note also that, the cache size of the Slaves is equal for both versions, and in the Lazy-Master version the Master node does not have cache because it does not need it.

In Figure 4(a), we can observe that up to a recordset of 4 millions of records, the results are almost the same in both versions. On the contrary, when the recordset exceeds 4 millions, the Worker-Master version is slower. As explained in Section 3 for the Worker-Master version, once the Slaves have finished performing the Sliding Window strategy over their block, they have to wait until the Master

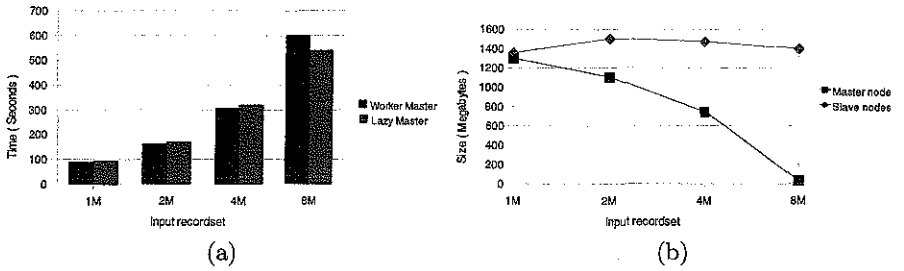

Fig. 4. Execution time for Worker-Master vs. Lazy-Master (a) and Cache sizes for the Master node compared to the Slaves nodes (b)

node finishes also with the Sliding Window over its own block. Afterwards, it sorts the recordset, divides the recordset into new blocks and sends them to the Slaves. In Section 2 the creation of the caches is described as a process that uses the maximum available memory. Since the Master node is the node that contains the input recordset, the larger the recordset size, the smaller the amount of available memory in the Master node. Figure 4 (b) shows that as the number of records grows, the Master cache size diminishes and the difference between the Slaves and Master cache increases. Therefore, the Master node takes longer in performing Sliding Window just because it has to do more comparisons, since they do not fit into its cache and the Slaves have to be waiting more time for the Master to finish.

This experiment supports the choice of using a single node, the Master, for containing the input recordset. It also shows that for large recordsets the Lazy-Master version is better. Since this paper is focused on the management of very large volumes of data, from now on we will work with the Lazy-Master version.

Time Analysis

In this experiment, we want to analyze the Record Comparison process time. This time is dissected as follows: the Detection time, which is the time spent in the Detection phase; the overhead time, which is the time spent in sending the different data over the network; and the Record Comparison preprocessing.

Figure 5 shows the dissection for the parallel version with 2, 4, 8 and 16 nodes. The input size has been fixed at 8 millions of records. Since we have parallelized the Detection phase, its weight over the total time decreases quickly while increasing the number of nodes, so this means that at some point, it is not useful to increase the number of nodes due to Amdahl's law [1]. Note that thanks to our technique, we have reduced the time to link 8 million records from 100 to only about 18 minutes.

It is also interesting to observe the influence of the overhead over the total execution time. The tests show that this overhead is negligible. Figure 6 presents

[1] Amdahl's law is used to find the maximum expected improvement to an overall system when only part of the system is improved.

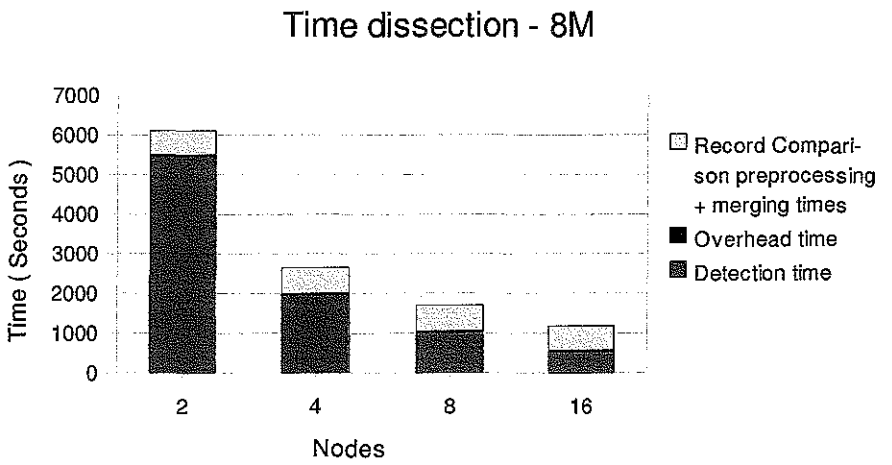

Fig. 5. Dissection of total time for 8M records

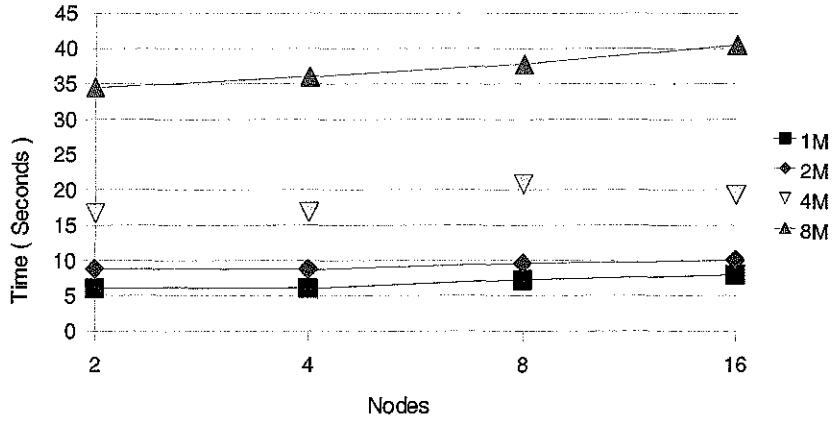

Fig. 6. Overhead produced by the parallelization

the value of the overhead time for the parallel version with 2, 4, 8 and 16 nodes. The input sizes are set at 1, 2, 4 and 8 million of records.

The most interesting observation of these results is the slow linear progression followed for each input recordset. This means that the proposed parallelization is very scalable in the number of nodes.

Speedup Analysis

In this experiment, we want to test the performance obtained with the parallelization proposed in Section 3. We use as baseline for calculating the speedup the sequential version of the Record Comparison process. The executions are run using from 2 to 16 nodes with 1 to 8 million records. We will also see the behavior when the 8M recordset is linked using an SMP parallelization with 2 threads.

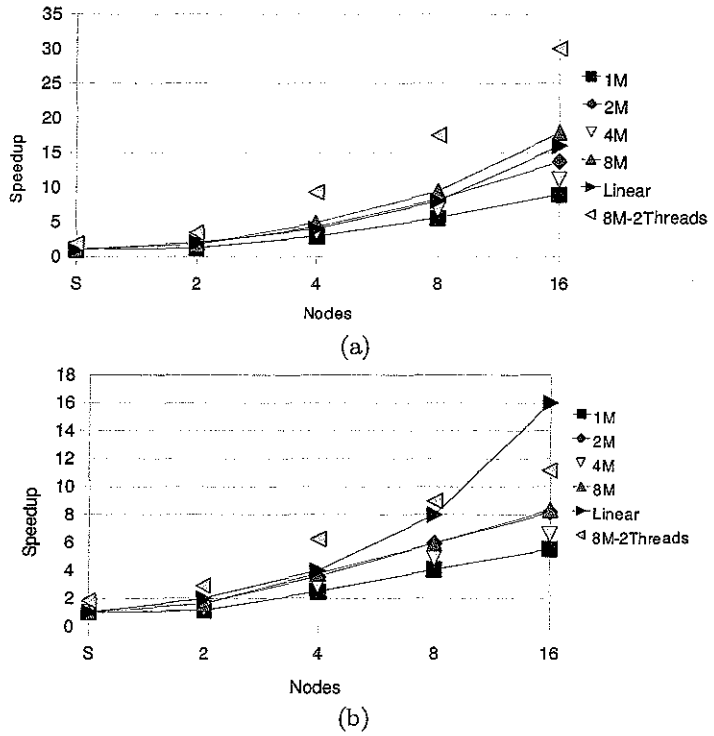

Fig. 7. Speedup for Detection phase (a) and Record Comparison process (b)

Figure 7(a) shows the speedup obtained in the Detection phase. We can observe that the speedup varies with the size of the input. Generally, the larger the recordset, the better the speedup. Since the differences are clearer in the case of using 16 nodes, we will focus on this scenario. These differences happen because in the sequential version, the larger the recordset, the smaller the caches that fit in memory. On the contrary, in the parallel version, the cache sizes of the Slaves are the same for any input recordset size, because the Slaves are not storing the input recordset. This explains the large distance between the speedup in the 1 million recordset case and the 8 million recordset case. To understand the short distance between 2 million and 4 million recordsets we have to take into account that, as explained in Section 3, since the cached values are no shared among Slaves, more comparisons are performed. Depending on the content of the recordset, the same comparison will be performed in more or less nodes and the speedup will be affected. Although these variations, we can observe that, in average, a significant speedup is obtained, and even better than linear for 8M records. It is really interesting to observe that the speedup for 8M-2threads doubles 8M using 1thread which proves that our distributed technique is complementary with an SMP based parallelization in each node.

We can observe the speedup obtained in the whole Record Comparison Process in Figure 7(b). Note that the speedup obtained is not linear because, as we have seen in the previous experiment, there is a constant time corresponding to the addition of Record Comparison preprocessing and merging time. However, we are able to perform the execution 8 times faster when we manage large recordsets using just 1 thread and 11 times faster when we use 2 threads.

5 Related Work

Re-identification methods are a specific class of data base techniques. These methods are designed to establish relationships among different entities or attributes stored in different data sources. Obtaining the relationships among entities or attributes makes sense in many scenarios such as: Schema matching [1], Data integration [13],Data cleaning [3] and Object integration [8].

There are different classic approaches for the reduction of work during the Record Linkage process, like the Standard Blocking [2,7] and the Sliding Window [10] methods, that intend to reduce the number of record comparisons. On the other hand, methods like RAR is aimed at reducing the number of attribute comparisons [14].

Finally, it is possible to find another approach to reduce the execution time of the RL process by using parallelism, as explained in [4], where it is necessary to know the state of the recordsets processed, either if they are clean or dirty. A good performance is achieved in [4] when the recordsets managed are clean. However, it is unsuitable for very large recordsets, specially when they are dirty, which is the common case. Note that in our approach we are not distinguishing between dirty and clean recordsets. In fact we are always assuming dirty ones since this is the worst case, assuming clean ones would mean do less comparisons among records, and therefore obtaining better times.

6 Conclusions and Future Work

In this paper we have shown that applying distributed strategies to a Record Linkage process is very useful and simple. This shows that organizations with little computing resources may use the PCs in their desktops in a Beowulf configuration to accelerate their RL processes in a cheap and efficient way.

Future work will include proposing more complex algorithms in order to increase the speedup, focusing on the preprocessing also, as the problem to tackle at this point. Another approach will be the analysis of clusters of non-homogeneous computing devices, where instead of dividing the recordset into blocks of equal size, it will be necessary to divide the recordset as a function of the available memory of the nodes, which will help us to obtain a better speedup. It will be also interesting to study the effect of sharing the cached comparisons among the nodes to reduce the amount of comparisons.

Another final approach will be to spread the input recordset out among the nodes, in order to be able to manage an input recordset that does not fit into the memory of a single node. This will imply also the use of parallel sorting techniques in the Detection phase.

Acknowledgments

Partial support by the Spanish MEC (projects ARES – CONSOLIDER INGENIO 2010 CSD2007-00004, eAEGIS – TSI2007-65406-C03-02 and TIN2006-15536-C02-02) and by the Government of Catalunya (grant 2005-SGR-00093 and GRE-00352) is acknowledged. Josep L. Larriba-Pey wants to thank UPC, Government of Catalunya and the Spanish MICINN for his I3 grant.

References

1. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. The Very Large Database Journal, 334–350 (2001)
2. Newcombe, H.B.: Record linking: The design of efficient systems for linking records into individuals and family histories. American Journal of Human Genetics (1967)
3. Do, H.H., Rahm, E.: COMA - A system for exible combination of schema matching approaches. In: Proceedings of the 28th Very Large Databases Conference, pp. 610–621 (2002)
4. Kim, H., Lee, D.: Parallel Linkage. In: CIKM, Lisboa, Portugal (2007)
5. `http://www.idescat.net`
6. Gómez, J., Larriba, J.L., Ribes, J.: Improving Record Linkage Performance. Technical report UPC-DAC-RR-2006-15
7. Jaro, M.A.: Advances in Record Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. Journal of the American Statistical Society, 414–420 (1989)
8. Atencia, M., Schorlemmer, M.: A formal model for situated semantic alignment. In: Proceedings of the 6th International Conference in Agent and Multiagent Systems (2007)
9. Bilenko, M., Basu, S., Sahami, M.: Adaptive Product Normalization: Using Online Learning for Record Linkage in Comparison Sopping. In: Proceedings of the 5th Int'l. Conference on Data Mining 2005, pp. 58–65 (2005)
10. Hernandez, M., Stolfo, S.: The merge/purge problem for large database. In: ACT SGMOD Conf. Proc., pp. 127–138 (1995)
11. Christen, P., Churches, T.: Febrl: Freely extensible biomedical record linkage. Joint Computer Science Technical Report TR-CS-02-05 (2002)
12. Brown, R.G.: Engineering a Beowulf-style Compute Cluster. Duke University Physics Department (2004)
13. Deen, S.M., Amin, R.R., Taylor, M.C.: Data integration in distributed databases. IEEE Transactions on Software Engineering (1987)
14. Sung, S.Y., Li, Z., Peng, S.: A Fast Filtering Scheme for Large Database Cleansing. In: International Conference on Information and Knowledge Management (CIKM), McLean, Virginia,USA (2002)

15. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady, 707–710 (1966)
16. Torra, V., Domingo-Ferrer, J.: Record linkage methods for multidatabase data mining. In: Information Fusion in Data Mining, pp. 101–132. Springer, Heidelberg (2003)
17. Winkler, W.E.: Data cleaning methods. In: Proc. SIGKDD 2003, Washington (2003)
18. Winkler, W.E.: Re-identification methods for masked microdata. In: Domingo-Ferrer, J., Torra, V. (eds.) PSD 2004. LNCS, vol. 3050, pp. 216–230. Springer, Heidelberg (2004)

Apèndix C

La distància de Levenshtein

En la teoria de la informació i en les Ciències de la computació, la distancia de Levenshtein és una mètrica que s'empra per mesurar quina és la diferència entre dues cadenes de caràcters. Donades dues cadenes de caràcters aquesta distància es calcula com el número mínim d'operacions per transformar una de les cadenes en l'altra. Les operacions que s'empren són la inserció, l'eliminació o la substitució d'un caràcter.

Es poden veure alguns exemple, com ara:

1 llegin → llegint (inserció d'una 't' al final de la cadena).

2 bariable → variable (substitució de la primera 'b' per 'v').

3 honada → onada (eliminació de l'h').

C.1 L'algoritme

La manera més freqüent de calcular la distància és mitjançant programació dinàmica. En l'Algoritme 1 es pot veure el pseudocodi que calcula la distancia entre la cadena $Str1$ de mida m i la cadena $Str2$ de mida n.

L'algoritme que calcula la distancia de Levenshtein empra una matriu de mida $(n+1) \cdot (m+1)$, on n i m són la mida de les cadenes de caràcters. La matriu es pot va omplint des de la part superior esquerra fins a la part inferior dreta, fila a fila. Per omplir una determinada posició (i, j) s'agafa el mínim entre la cel·la superior $((i-1, j))+1$, la cel·la de l'esquerra $((i, j-1))+1$, i la cel·la que es troba en la diagonal superior esquerra $((i-1, j-1))+cost$, on $cost$ és igual a 0 si, l'i-èssim caràcter de $Str1$ és igual a el j-èssim caràcter de $Str2$, altrament $cost$ igual a 1 (línies 7-11). Cada salt horitzontal (línia 13) o vertical (línia 14) correspon a una inserció o a una eliminació respectivament, un salt diagonal (línia 15) correspont a una substitució si $cost$ igual a 1, i altrament correspon a no fer res. D'aquesta manera, en finalitzar l'algoritme, la distància de Levenshtein entre $Str1$ i $Str2$ està emmagatzemada en la posició (n, m) de la matriu.

Algoritme 1 Distància de Levenshtein

```
1: funció LevenshteinDist(char Str1[1,···,m], char Str2[1,···,n])
2:     int d[0···m,0···n]
3:     int i, j, cost
4:
5:     per tot i ∈ 0, ···, m fer
6:         per tot j ∈ 0, ···, n fer
7:             si Str1[i] = Str2[j] aleshores
8:                 cost:=0
9:             altrament
10:                cost:=1
11:            fi si
12:            d[i, j] = min(
13:                            d[i − 1, j  ] + 1,          ▷ Eliminació
14:                            d[i  , j − 1] + 1           ▷ Inserció
15:                            d[i − 1, j − 1] + cost      ▷ Substitució o no fer res
16:                        )
17:        fi per
18:    fi per
19:    retorna d[m, n]
20: fi funció
```

La Figura C.1 representa la matriu resultant de la comparació entre "cervell" i "ser bell".

En la Figura C.1 s'observa que la distancia entre "cervell" i "ser bell" és de 3. La matriu resultant també indica, mitjançant les fletxes de colors, les transformacions que s'han de fer per transformar "ser bell" en "cervell" en el nombre mínim de passos. S'observa que per dur a terme la transformació hi ha dos camins mínims. Les fletxes vermelles marquen el tros de camí que és conjunt mentre que les blaves i verdes indiquen en que es diferencien els dos camins.

Si s'interpreten les fletxes de la matriu, com els salts descrits en l'Algoritme 1, es veu que les dos camins per transformar "ser bell" en "cervell" són els de la Figura C.2.

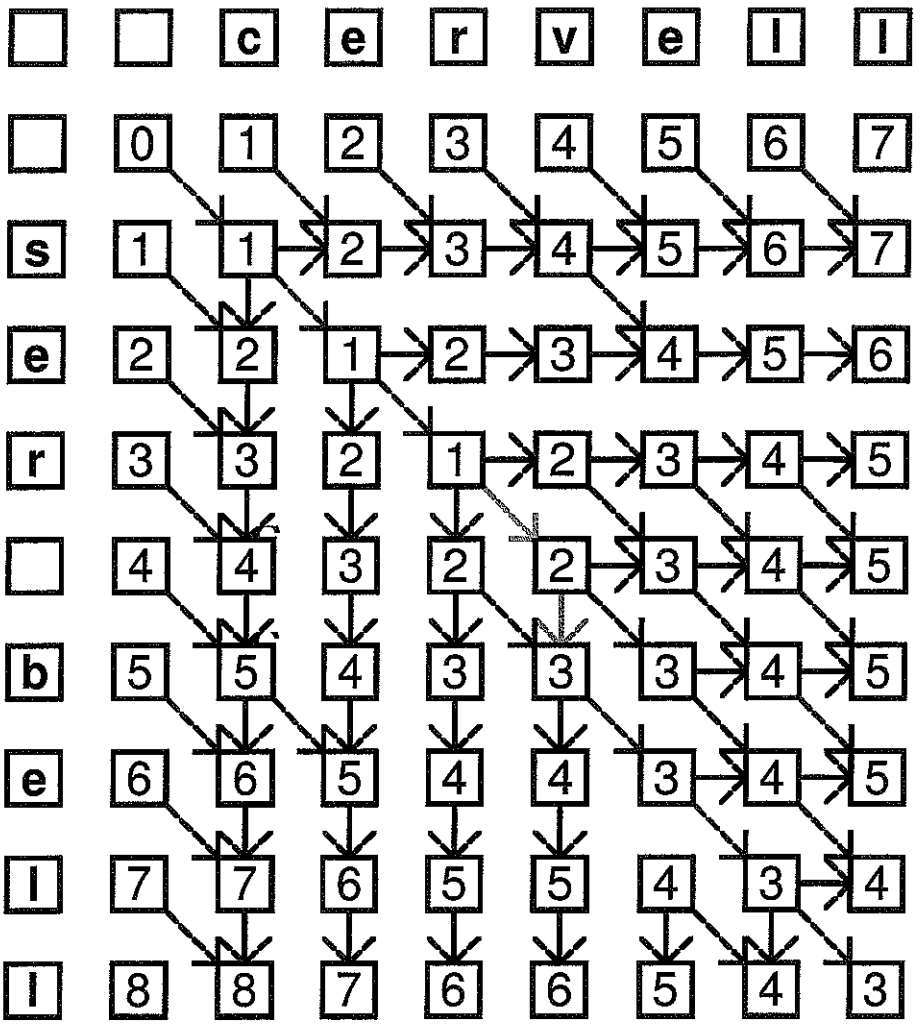

Figura C.1: Matriu que resulta de l'Algoritme 1 amb Str1="cervell" i Str2="ser bell".

camí blau

s	e	r		b	e	l	l
's'->'c'	=	=	-	'b'->'v'	=	=	=
c	e	r		v	e	l	l

camí verd

s	e	r		b	e	l	l
's'->'c'	=	=	' '->v	-	=	=	=
c	e	r	v		e	l	l

Figura C.2: Possibles transformacions per transformar "ser bell" en "cervell".

Bibliografia

[1] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. Dins *AFIPS Conference Proceedings*, volum 30, pàgines 483–485. 1967.

[2] Manuel Atencia i W. Marco Schorlemmer. A formal model for situated semantic alignment. Dins Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns i Onn Shehory, editors, *AAMAS*, pàgina 226. IFAAMAS, 2007.

[3] RA Baeza-Yates. Searching the Web: challenges and partial solutions. Dins *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pàgines 23–31. 1998.

[4] R. Baxter, P. Christen i T. Churches. A comparison of fast blocking methods for record linkage. Dins *ACM SIGKDD*, volum 3, pàgines 25–27. 2003.

[5] A. Beguelin, J. Dongarra, W. Jiang, R. Manchek i V. Sunderam. *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT Press Cambridge, MA, USA, 1995.

[6] R.G. Brown. Engineering a Beowulf-style Compute Cluster. *Physics Department, Duke University*, 2004.

[7] D.R. Butenhof. *Programming With Posix Threads*. Addison-Wesley Professional, 1997.

[8] R. Chandra. *Parallel Programming in OpenMP*. Morgan Kaufmann, 2001.

[9] P. Christen, T. Churches, Australian National University Computer Sciences Laboratory i Australian National University Dept. of Computer Science. *Febrl-Freely Extensible Biomedical Record Linkage*. Australian national University, Dept. of Computer Science, 2002.

[10] SM Deen, RR Amin i MC Taylor. Data Integration in Distributed Databases. *Transactions on Software Engineering*, pàgines 860–864, 1987.

[11] H.H. Do i E. Rahm. COMA: a system for flexible combination of schema matching approaches. Dins *Proceedings of the 28th international conference on Very Large Data Bases-Volume 28*, pàgines 610–621. VLDB Endowment, 2002.

[12] J. Gomez-Bao, N. Martinez, F. Escale, J. Ribes-Puig, V. Muntes-Mulero i J-L. Larriba Pey. Memoization techniques to improve Entity Resolution performance. Report tècnic, Universitat Politècnica de Catalunya - Departament d'Arquitectura de Computadors, 2007.

[13] W. Gropp, E. Lusk i A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, 1999.

[14] M.A. Hernández i S.J. Stolfo. The merge/purge problem for large databases. Dins *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pàgines 127–138. ACM New York, NY, USA, 1995.

[15] D. Jiménez-González, J.L. Larriba-Pey i J.J. Navarro. Communication conscious radix sort. Dins *Proceedings of the 13th international conference on Supercomputing*, pàgines 76–82. ACM New York, NY, USA, 1999.

[16] D. Jiménez-González, J.J. Navarro i J.L. Larriba-Pey. Fast parallel in-memory 64-bit sorting. Dins *Proceedings of the 15th international conference on Supercomputing*, pàgines 114–122. ACM Press New York, NY, USA, 2001.

[17] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Dins *Soviet Physics Doklady*, volum 10, pàgina 707. 1966.

[18] C. LOSER, C. LEGNER i D. GIZANIS. Master Data Management For Collaborative Service Processes. Dins *International Conference on Service Systems and Service Management, Beijing, July*, pàgines 19–21. 2004.

[19] M. Michalowski, S. Thakkar, C.A. Knoblock i UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES INFORMATION SCIENCES INSTITUTE. *Exploiting Secondary Sources for Unsupervised Record Linkage*. Defense Technical Information Center, 2004.

[20] Erhard Rahm i Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

[21] W.E. Winkler. Data cleaning methods. Dins *Proc. SIGKDD*. 2003.